UNIVERSITY OF OSLO
Department of informatics

Multiobjective Optimization of an Ultra Low Voltage/Low Power Standard Cell Library for Digital Logic Synthesis

Master thesis
60 credits

Martin Severin Orrebakken Haugland

15.05.2012

Abstract

In this thesis there has been construct a standard cell library in 90nm CMOS technology meant for scientific and/or medical research purposes.

This library is meant to operate on a supply voltage of 300mV, for example in order to improve the uptime of battery and solar cell powered devices, as well as be compatible with alternative power sources (heat, vibration, induction, etc). The performance variables for the library components were optimized with respect to delay, static power dissipation and robustness for ultra low voltage (ULV) operation. This has been done with a Multiobjective Optimization approach.

The cells that have been optimized is a NOT, a NAND and a NOR gate. They have been used to design an XOR gate, a DFF and a D-LATCH. The three optimized cells and the XOR have been completed for synthesis. From these four cells, all types of digital logic can be synthesized. This has been shown with the synthesizing of a 32-bit adder. The 32-bit adder has been tested on 300mV and over several temperatures.

Although operating circuits at low supply voltages offers advantages in terms of power and energy consumption, this method also introduces several problems such as increased delay variations and worsened noise margins.

The simulations were preformed in a commercial 90nm process with high threshold cells. This was provided by TSMC. The simulations were carried out in Cadence Virtuoso Platform, Cadence Encounter RTL Compiler (encounter), Cadence Virtuoso Spectre and MatLab.

Preface

This thesis is a part of the degree *Master of Science* in Nano-and Microelectronics at the Department of Informatics (Ifi), Faculty of Mathematics and Natural Science, University of Oslo (UiO). The project was started in May 2011 and concluded in May 2012.

Working with the thesis has been both interesting and challenging. The work has provided me with valuable knowledge about among other things: ultra low voltage operation, multiobjective optimization, Pareto optimality, CMOS design methods, simulation methods and library characterization.

First and foremost I want to express my gratitude to my supervisor, Snorre Aunet, and my co-supervisors, Amir Hasanbegovic and Hans Kristian Otnes Berge. They have given me valuable guidance, technical support and have been an important source of inspiration and knowledge.

I also want to thank the other master students at the computer lab, Tor-Eivind, Morten, Geir and Michael, for both technical discussions and quite off topic conversations, as well as several sets of billiard.

Thanks also to my mother-in-law for her thorough proofreading on a subject that is “somewhat” different from her usual high school papers.

Last, but not least, I want to thank my family for their support throughout my studies and master’s thesis, and I want to give special thanks to my girlfriend, Siri, for her continuing support and for helping me improve the language of the thesis.

Contents

List of figures

List of tables

Chapter 1

1 Introduction

During the last years there has been an increased focus on expanding the uptime of battery powered devices [1] or to have solar powered devices. Since battery energy density is moving towards a maximum [2], alternative methods should be considered, hence reducing the power consumption on a chip. One of the most effective ways of reducing power in a digital chip is to reduce the supply voltage. Other approaches have been smaller transistors with constant field scaling and near threshold circuits [3]. The problem with CMOS scaling is that it seems to be coming to an end [4]. Earlier technologies have always had a clear successor. The CMOS has, as of this moment, not a clear successor. We have, in this work, focused on circuits operating on a supply voltage lower than the transistors inherent threshold voltages.

1.1 Motivation

A standard cell library is a collection of cells that can be synthesized to a larger design, which is described with a hardware description language (HDL). Standard cell libraries are used for a large range of applications. One of the more common uses of a standard library is within the design of application-specific integrated circuits (ASIC). The use of a standard cell library drastically reduces the cost of designing a chip. It also reduces the time-to-market (TTM), which results in lower production cost, early sales, longer time in marked, etc.

Smaller companies or companies specialized in a specific field do not necessarily have the broad knowledge that the large ASIC companies have (e.g. a company specializing in analog design may not need to use much resources on the digital design when this can be synthesized to work optimally with the analog circuits).

Full custom design enables you to get most out of the design. Custom design takes time and it is expensive. The newest custom chips today can take several hundred to several thousand man-labor years to produce. If one is to produce a larger concept prototype or a large chip for a few cases, this will be too costly. This is why standard cell libraries are used.

Synthesizing tools can, together with a proper defined standard cell library, synthesize chips for production, concept testing and debugging. Several companies also use something called semi-custom design. This is used if the company has customized larger sections of the design, but uses them in a library with a synthesizing tool to create the large design (e.g.

custom memory cells), or if they synthesize the least critical path with a standard cell library and customize the critical areas.

The goal of this thesis is to produce a standard cell library that operates at subthreshold voltages for synthesizing prototypes and devices for medical and scientific purposes.

1.2 Previous work

In the 1980s a designer had to choose an ASIC manufacturer and implement his design using the design tools from the manufacturer. Although there were third party tools in the marked, there was no link between the design tools and the manufacturers. The solution to this problem was the realization of the standard cells. ASIC manufacturers could now create functional blocks with known electrical characteristics.

The idea of minimizing power dissipation in cells by lowering the supply voltage has been known for decades. As the technologies have moved from one node to another, the feature sizes have moved down and the supply voltage has moved down accordingly. In the same time the manufacturers have created new libraries with standard cells for the new sizes and voltages. Libraries for specific purposes have also been created, for example RF technology, high speed circuits, radiation tolerant circuits or circuits with a low power consumption (these are often custom and/or Intellectual Property (IP) libraries [5]). This is because the different problems have different objectives and solutions. A library for space chips might have larger demands towards radiation tolerance than a library for medical implants. Chips for medical implants might, on the other hand, have a demand for low power consumption. There is, however, no manufacturer that, to our knowledge, has made a standard cell library for sub threshold. In [5], Khosrow Golshan explains the essential steps needed for the design of an ASIC, this includes information about library design.

Christian Piguet has in [6] collected the writings of several authors that write about design of low-power circuitry. The book covers several of the low-level aspects of the design of low-power integrated circuits (ICs), and some of this can be used in the design of a subthreshold standard library.

In [7], Blesken, Lükemeier and Rückert show the need of designing a cell library in the subthreshold region and describe how they have used multiobjective approach on standard cells.

1.3 Overview of the thesis

This thesis will go through the optimization process of a low voltage/low power library. The library is set to work on 300mV and is designed with cells that have 450mV threshold voltage (V_t). The optimization has been done, using multiobjective optimization techniques with Pareto optimality. This technique is used for optimizing towards low current leakages, low delay and high robustness.

The optimized library cells, a NAND, a NOR and a NOT, will be used for Very-large-scale Integration (VLSI). These cells have also been used to design an XOR, a DFF and a D-LATCH.

For concept testing, a 32-bit adder has been synthesized and tested to work properly on 50 kHz using only 60fJ.

The chapters and appendixes contain the following:

- Chapter 1 presents the motivation for working with and the manufacturing of a low power library, and it talks about work done on similar topics.
- Chapter 2 gives an introduction of the background of circuit design, low voltage design, current leakage and power dissipation.
- Chapter 3 presents the simulation and optimization process and explains choices that had to be taken in order to construct the library.
- Chapter 4 presents a summary of the findings from the thesis, as well as a proof of concept with a synthesized 32-bit adder.
- Chapter 5 discusses some of the main aspects of the thesis.
- Chapter 6 summarizes the main contributions, and lists some ideas and suggestions for future work.
- Appendix A explains more detailed the layout methodology and rules connected to layout design.
- Appendix B shows the design flow of the work that has been done.
- Appendix C shows schematic drawings and truth tables for the library cells.
- Appendix D contains the code for the grid based multiobjective optimization and all the simulation files for the NOT gate.
- Appendix E shows the layout view of the library cells.
- Appendix F shows the simulation graphs from the NOT gate and the 32-bit adder that was synthesized.
- Appendix G shows the schematic view of the full 32bit adder design.

Chapter 2

2 Background

During the last decades the struggle between power dissipation, timing performance and complexity of a device has dominated chip design. New technologies that are quicker, have a better noise margin, a lower supply voltage, a smaller area per function and use less power per calculation have been the main focus for increasing a chips' efficiency. Over the last decades, the number of transistors on a chip have increased exponentially in accordance to Moore's law [8].

When optimizing a circuit, an objective of the optimization may be to have low static power dissipation in the cell. Since it seams that we are moving towards the scaling-limit of CMOS technology, the focus should taken over to separate objectives in stead of focusing on several. This tells us that we have to focus on key performances for a design. This means that in order to for example reduce timing or increasing the energy efficiency, other performance variables must be sacrificed. When in the same technology node it will become harder, or close to impossible to satisfy all performance variables possible. Alternative to improving selected performance variables, new technologies with other materials can be considered since this has shown some advantages over standard CMOS [9].

When we look at the history of the technologies, from vacuum tubes to NMOS-based technology, earlier technologies have been replaced by their successors when their energy overhead became prohibitive. There is however, no clear successor for the CMOS today [10].

2.1 Low voltage operation

Several ways of reducing the power consumption have been used or tested during the last decades. The traditional way has been to move from one technology and over to a newer one with smaller gate sizes. When reducing the gate sizes it has in the last technology nodes also been normal to reduce the power supply. This is known as constant field scaling [3]. In the last technologies the transistor sizes have become so small that current mismatch due to local doping fluctuations has become a major issue [11, 12]. Since we are closing in on the limit of CMOS [4], new ways of reducing energy must be considered.

Another method to reduce the power consumption on a chip has been to reduce the supply voltage on circuits that do not necessarily need the highest speed. Subthreshold operation refers to using a supply voltage that

is less than a single transistors threshold voltage [1]. Near threshold operation refers to using a supply voltage that is close to or slightly above the transistors threshold voltage [13]. Reducing the supply voltage on a standard cell library will give lower power consumption, but it will be even lower if the library is designed for a specific supply voltage [7, 14]. Several estimations of the theoretical lowest limit on the supply voltage have been given, but a digital circuit should be able to run as low as a 100mV ("$4U_T$" at ambient temperatures)[15].

2.1.1 Subthreshold

Subthreshold logic operates in the weak or moderate inversion region, which is the area of operation that is defined as the region where the power supply is below the transistors threshold voltage (V_t). This is one of the ways to get lower power consumption [16]. In theory, digital circuits can be operated at a supply voltage as low as 100 mV [15].

Operating in the subthreshold region of the transistor gives us a near ideal transfer characteristic, low gate leakage and an input capacitance that is lower than that of strong inversion. But in addition to this, operating in subthreshold also leads to some major challenges concerning robustness [17, 18]. This means we might have to accept a higher failure rate than what normally would be considered acceptable. Alternatively it will be necessary to use a more costly and special designed process. Another option is to operate in a higher region of the subthreshold region. This will reduce the failure rate considerably [1].

2.1.2 Near threshold

Near threshold logic is one way of getting some of the positive effects of both areas. Near threshold operations on customized library cells should therefore have a better robustness than subthreshold cells and lower power consumption than standard cells with reduced power supply. In [18] they show that 20% increase in energy from minimum energy point can give back up to ten times in performance. This is because standard cells are designed with minimum lengths, and it is shown in [19] that there can be advantage to vary both lengths and widths to get higher performance on lower voltages.

2.2 Digital CMOS circuits

Digital circuits [20-22] are made from analog components. E.g. Digital circuits constructed in CMOS are made out of nMOS and pMOS transistors, which are analog components. In an ideal component there would be zero rise time and unlimited fan out. Since the circuits are not ideal in real life, the design must assure that the unwanted analog effects do not dominate the design.

Digital cells have as function to transform one of more input signals to one or several output signals according to a truth table. In digital logic there are several types of gates that operate differently. These can be divided into two main groups: combinatorial and sequential logic.

2.2.1 Combinational logic

A digital circuit that directly transforms the input logic values to give a single output logic value, according to a Boolean function, is considered a combinatorial logic circuit. An example of this is the NAND gate that, according to the truth table in Appendix C, only relies on the values on its inputs I1 and I2 to give a value to the output. Combinatorial logic can also be more complex. As long as the circuit represents a Boolean function it is a combinatorial logic that presents the truth table. An example of this is a 4-to-1 multiplexer:

$$F = (A \cdot \overline{S_0} + B \cdot S_0)\overline{S_1} + (C \cdot \overline{S_0} + D \cdot S_0)S_1 \quad \textbf{(2-1)}$$

The output F will always get an answer when we put in values for A, B, C, D, S_0 and S_1. A combinatorial logic cell will always present the output value according to the truth table, without the dependency of a clock.

2.2.2 Sequential logic

A sequential logic relies, as the combinatorial logic, on a truth table with fixed inputs. The difference is that sequential logic also relies on the history of the inputs. The sequential logic therefore has a state memory. The D-latch is an example of a sequential circuit. As shown in the truth table in Appendix C, the output can be dependent on the previous output. Other examples of sequential logic are some types of computer memories, Moore and Mealy State machines.

2.3 Leakage current and power dissipation in digital CMOS

CMOS has during the past decades emerged as the main technology in VLSI circuits. One of the reasons for this is the power consumption characteristic. In standard CMOS technology operating at a supply power well over the transistor's threshold voltage, V_t, major power consumption only occurs in the switching between logic states.

In a digital cell there are three types of power dissipation that must be taken into account: dynamic, static and short-circuit power dissipation. The total power dissipation can therefore be expressed as a sum of these components [22].

$$P_{total} = P_{dynamic} + P_{static} + P_{short\ circuit} \quad \textbf{(2-2)}$$

Static power dissipation and short-circuit power dissipation in traditional CMOS have usually been ignored since the dynamic power dissipation has dominated the total power dissipation. In newer CMOS technologies the increased leakage and the decreased dynamic dissipation make it a larger part of the total power dissipation [23]. The static and short-circuit power dissipation must, as a result, be taken into account when calculating the total power dissipation.

2.3.1 Dynamic power dissipation

Dynamic power dissipation is a product of activity rate and charging and discharging of load capacitance (and parasitic capacitance) when transistors are turned on and off. The charging and discharging of the output capacitance is usually the dominant term. The energy stored in a capacitor when charged from 0 to V_{DD} is:

$$E_c = \frac{1}{2} C V_{DD}^2 \quad \textbf{(2-3)}$$

The dynamic power dissipation can therefore be written as:

$$P_{dynamic} = C_L V_{DD}^2 f_p \quad \textbf{(2-4)}$$

where C_L is the average load capacitance, V_{DD} is the supply voltage and f_p is the repetition frequency [22]. The repetition frequency is the sum of the activity factor and the clock frequency or the average frequency of the switching.

2.3.2 Static power dissipation

Static power dissipation is due to leakage current or current drawn constantly from the power supply [22].

$$P_{static} = I_{static} V_{DD} \quad \textbf{(2-5)}$$

In a complimentary circuit, half of the transistors will have transition between source and drain most of the time. Take a NOT gate, the pMOS will have transition between source and drain if the gate is high and the nMOS if the gate is zero (close to ground). When a transistor has no transition between source and drain, there will be some parasitic effects between gate, source and drain that are explained in Section 2.3.4. Historically, static power dissipation was small compared to the dynamic power dissipation. It has therefore not been a large part of the power dissipation calculations. When the supply voltage is moving downwards, it will reduce the dynamic power dissipation and the static power dissipation will become a larger part of the total power dissipation.

2.3.3 Short-circuit power dissipation

Short-circuit power dissipation is due to the short circuit current from power supply to ground, this happens in a brief period during the transition between logic states when both the pMOS and nMOS transistors are on [24]. When the dynamic power dissipation is going down, the Short-Circuit power dissipation will become a larger part of the total.

2.3.4 Leakage sources

In nano-scaled CMOS circuits, there are many leakage sources. The most important sources are gate leakage, subthreshold leakage and the reverse biased junction BTBT leakage [25].

Subthreshold leakage current (I_{SUB}) is the most dominant among the

leakage sources. I_{SUB} became an issue at the 180 nm technology and became a problem in smaller technologies (90 nm, 65 nm, etc) [26].It is caused by minority carriers drifting between source and drain when the transistor is operating in the cutoff region (see Figure 2-1).

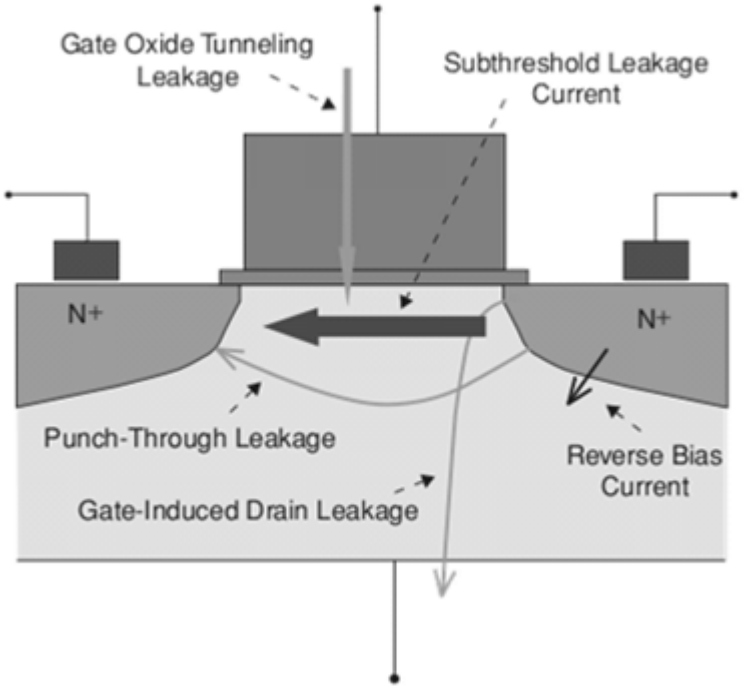


Figure 2-1: Illustration of leakage currents [26]

Gate leakage current (I_G) is an element that becomes a serious concern in transistors with a gate thickness below 2 nm. With thin gate oxide, small potential differences can cause a high electric field which in turn causes tunneling through the oxide. The total gate leakage is given as:

$$I_G = I_{GC} + I_{GB} + I_{GS} + I_{GD} \quad \textbf{(2-6)}$$

BTBT current leakage represents the reverse biased p-n junction from drain/source to the bulk. BTBT leakage tends to be significant in 65nm technologies and below. By increasing the lengths from the minimum, one can reduce this effect.

Chapter 3

3 Simulation theory and method

One of the most important roles of design and optimization work, is to make sure that the circuits and the systems operate safely and according to specifications.

In order to design an efficient CMOS library for integrated circuits (IC), it is possible to solve optimization problems of more than one objective. Multiobjective optimization is important because there is rarely one ideal solution that provides the best result for the problem [27]. Multiobjective optimization is the process of optimizing several conflicting objects simultaneously. For larger designs, this optimization is easiest done on the smallest unit: the standard cell. This will improve the performance of the entire design [28, 29]. In order to do this when designing a library, we must first decide which circuits to design and simulate on. This means deciding what standard cells to have in the library and set layout rules, explained in Appendix A. The schematic of the cells is shown in Appendix C, and the optimization algorithms used to find the values for the cells is attached in Appendix D.

The classical way of designing CMOS logic is to get a symmetrical DC curve $V_{in} \rightarrow V_{out}$ which gives reasonable values in noise margin and time, but not necessarily in all objectives [28].

It is shown in [7] that the relative noise margin NM/V_{dd} of a set of CMOS NOT gates in a commercial standard cell library decreases when the supply voltage is lowered. In [30], it is shown that the lengths and widths affect the threshold voltage. This means that it can be advantageous to vary all sizes on the transistors when designing a digital cell for low power.

When comparing two circuits with the same function, but different performance variables, it is important to have a good set-up for the testing and a clear idea of what we are optimizing towards.

Since optimizing can be a computationally demanding process, it is valuable to simplify the optimization and design in order not to use more time and computational power than we have at our disposal. Even with reduced complexity, it might take too much resource to simulate the cells. Although PC hardware has improved a lot, simulations can still be time consuming on today's most powerful computers [31]. Parallel computing is therefore an option that must be considered.

When creating layout for a library, it is also important to know the design rules of the process and the design tools in order to design a cell

that is accepted. The workflow is shown as a diagram in Appendix B.

3.1 Multiobjective optimization

When sizing transistors the focus should be on energy efficiency, robustness and speed. These are a product of the transistors lengths and widths as well as process' parameters. After choosing a process, it is not possible to change the process given parameters. The only parameters we can change are the lengths and widths of the transistors. This gives us simulations that can have up to 2×M dimensional inputs, where M is the number of transistors in a design. Since this can result in very large simulations, it can in certain occasions be of advantage to simplify the designs.

The traditional way of designing standard cells has been to use minimum lengths for the transistors and only vary the widths. This is to reduce the input capacitance. As shown in [32], when not working with superthreshold logic, it can be advantageous to use other lengths than minimum lengths to improve the circuit parameters.

When a problem can be quantified, equations can be used to get a numerical value on how well different objectives perform, these values can be compared and optimized. Optimization is an essential process when designing systems and is used in many areas, including mathematics, physics (e.g. electronic design and simulation on magnetic induction) and informatics (e.g. chip design, timing analyses, and mathematical approaches). The objective is to simulate a matrix of performance variable and see which combination that works best. All the objects have maximum limits that must be held, but as long as no one of the objects exceeds their limit the result will be considered. An optimization problem with N objectives F_1, F_2, …, F_N can be expressed as:

$$\text{Min} \qquad F_n(x), n = 1,2, \dots N, \qquad \textbf{(3-1)}$$

where x is the vector of decision variables. The solution to the above problem is a set of Pareto points. As a result, instead of a unique solution to the problem, one gets a possibly infinite set of Pareto points.

3.1.1 Pareto efficiency of a Multiobjective problem

In an optimization algorithm containing more than one solution, there will be conflicting objectives. In most cases it will be next to impossible to satisfy all the objectives (e.g. it is impossible to have the fastest and most robust logic cell with the smallest amount of power dissipation and smallest cell area). The concept of Pareto optimality is to find the Pareto front for a multiobjective problem. In an X-dimensional (X-D) solution, where X is the number of objectives, all the Pareto efficient points make up an X-D Pareto front. In a system that has several simulated points with an X and Y values, a solution that cannot be better in either the X or Y direction without getting worse in the other, is Pareto efficient.

In order to test if a result is Pareto efficient, the conditions of a simulation have to be changed and the new result has to be compared to the old one. Figure 3-1 shows an example of a Pareto front. All the squares

represent simulated results. The smaller values will be preferred to larger. In this example, C is not a Pareto point because it is dominated by both A and B. If the result is not dominated by another solution in the search space, like A and B is, it is a Pareto optimal solution [33].

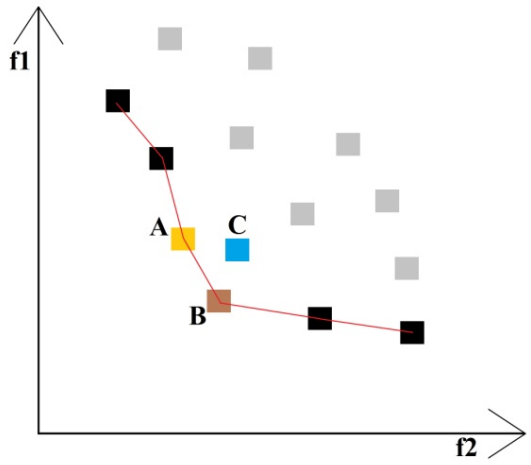


Figure 3-1: Example of a Pareto frontier

Since the Pareto front will have as many dimensions as the calculations have objectives, it can be difficult to present the front in a proper manner. One good way of presenting the result is to plot a series of connected 2-D plots where the points will light up in all plots when chosen. In a multi object Pareto front, it will be up to the designer to decide which one of the points along the Pareto front to choose. If the designer sets up minimum and maximum criteria for each objective, the Pareto Frontier will become smaller and easier to read. If there are several points that are within the same area, the designer can also decide which one to use based on the input values (if lengths and widths are optimized, there will be some circuit sizes that are easier than others to realize).

3.1.2 MatLab Optimization Tool Box

One of the MatLab tools that can be used is Optimization Tool Box with Genetic Algorithm Multi Objective Optimization ("*gamultiobj*" optimization) [34]. This tool simulates by using selected variables (random at first run, then selected by the genetic algorithm after analyzing the answer from the random/previous runs). The main goal of this tool is to reduce the value of the outputs of the fitness function (see Appendix D.4). By changing the input variables the tool searches for a optimal Pareto front. The output of the tool is an N dimensional matrix with Pareto points, where N is the number of different objectives.

3.1.3 Grid based multiobjective optimization method

Typical multiobjective optimization (MOO) algorithms, particularly *gamultiobj*, expect continuous input parameters. However, CMOS circuits are in general limited to a fixed design grid, thus only discrete parameter values are allowed. This means that algorithms, such as *gamultiobj*, can test many points close to a legal discrete point, while only the exact

discrete points are of interest. To rectify this, we employed a grid-based algorithm, included in Appendix D.3.

The algorithm starts by creating points in an N-dimensional grid, where N is the number of variables. For a simulation with four variables where we want to compute the objective functions for all points, this will produce a huge amount of computations. If all the four variables have a list of 30 values, this will result in 810 000 (30^4) variable set.

In order to reduce the number of calculations we start with a limited size grid, from which the objective function of all points are computed. In a second step, the algorithm then discards dominated points and keeps the points currently in the approximated Pareto front. The algorithm will then use these points to compute the next set of candidates, example shown in Figure 3-2. These candidates are located in steps of one half of the previous grid step, in all possible directions and combinations of directions. Based on the previous front and the new points a new approximated front is computed and the algorithm repeats from the second step until a suitable minimum step has been achieved.

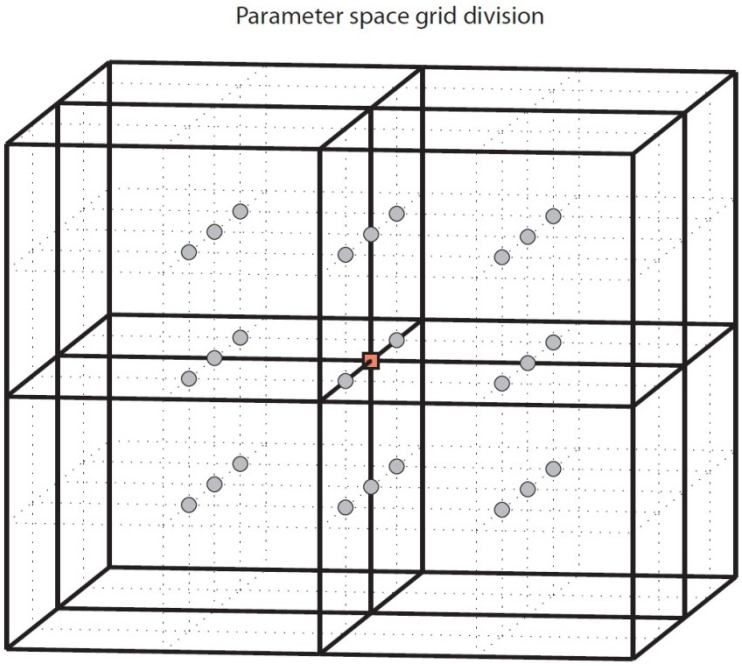


Figure 3-2: Example of new computed points for optimization

3.2 Objective parameters

The simulations were set to rely on only DC simulation, this was because the transient simulation added several seconds per simulation. The DC simulation will find static power but not be able to tell us anything about the dynamic power. This is explained in more detail in Section 3.5.

In the simulations all the results are calculated from the I_{off} and I_{on}. When running the simulations the I_{off} and I_{on} will be measured at the edges of the noise margin. The I_{off} and I_{on} at the noise margin will be called I_{off}* and I_{on}*. This is to avoid mixing the terms.

3.2.1 Noise margin

Noise margin is a parameter related to the input-output voltage characteristics. The noise margin allows us to decide the acceptable amount of noise voltage on the gate input so that the output is not affected. There are commonly two parameters used to specify the noise margin, LOW noise margin, NM_L, and HIGH noise margin, NM_H [35].

If one has a logic gate, in this case a NOT gate, which satisfies the relationship:

$$V_{out} \geq V_{OH} \; when \; V_{in} \leq V_{IL} \quad \textbf{(3-2)}$$

$$V_{out} \leq V_{OL} \; when \; V_{in} \geq V_{IH} \quad \textbf{(3-3)}$$

then one can define the NM_L and NM_H by the definitions:

$$NM_H = |V_{OH} - V_{IH}| \quad \textbf{(3-4)}$$

$$NM_L = |V_{IL} - V_{OL}| \quad \textbf{(3-5)}$$

where:

- V_{IL}=Highest input considered as logical zero
- V_{IH}= Lowest input considered as logical one
- V_{OH}= Lowest output considered as logical one
- V_{OL}=Highest output considered as logical zero

This is illustrated in Figure 3-3 and Figure 3-4.

In general it is desirable to have $V_{IH} \geq V_{IL}$. This would mean that there would be sufficient gain in the transition region. A logic combinatorial gate is self-consistent when any possible input taken from the logic range produce an output state that lies in the correct logic range [36].

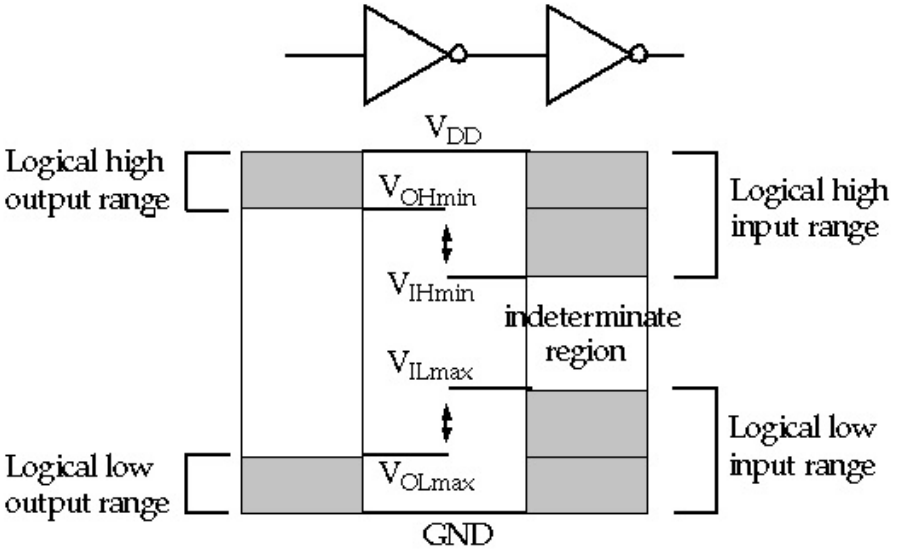


Figure 3-3: Illustration of the noise margin definition

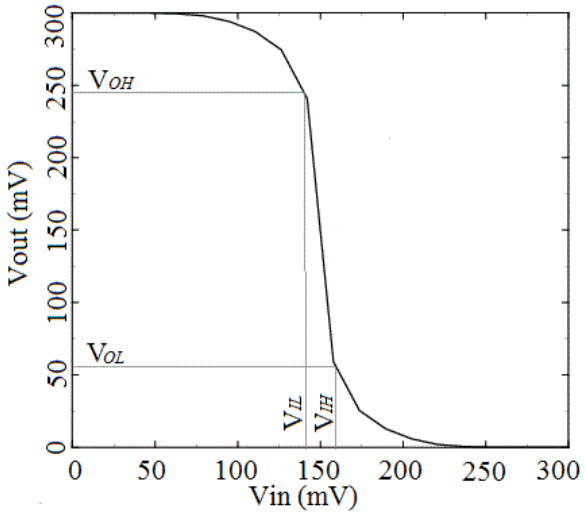


Figure 3-4: Illustration of an ideal CMOS NOT gate

3.2.2 Robustness

The robustness of a circuit describes the circuit's ability to operate under changing conditions, such as different temperatures. There are several ways of measuring the robustness of a cell, but there is no fixed standard. In a statistical test, a circuit with high robustness will have a low error probability. If we define an error to be the case when a circuit fails to satisfy the noise margin criteria of the preceding section, we will also have to satisfy the following:

$$Ion_n > Ioff_p \text{ when } V_{in} >= \text{NMH and } V_{out} >= NML \qquad \textbf{(3-6)}$$

$$Ion_p > Ioff_n \text{ when } V_{in} <= \text{NML and } V_{out} <= NMH \qquad \textbf{(3-7)}$$

This tells us that the I_{on} must be higher than the I_{off} and to optimize robustness we must maximize the I_{on} to I_{off} ratio. In subthreshold, the I_{on} and I_{off} may vary considerably from transistor to transistor particularly, due to random doping fluctuations [11]. This may cause I_{on} to be lower than normal, while I_{off} is higher. We must therefore consider this variation for the ratio. The distribution for I_{on} and I_{off} are almost log-normal, so if we would like to express the robustness of a circuit in terms of a design margin as μ/σ we get:

$$\text{Robustness } \left({}^{\mu}/_{\sigma}\right) = \frac{\text{mean}\left(\log_{10}\left(\frac{I_{on}*}{I_{off}*}\right)\right)}{\text{std}\left(\log_{10}\left(\frac{I_{on}*}{I_{off}*}\right)\right)} \qquad \textbf{(3-8)}$$

This equation is used to evaluate robustness in the simulations shown in chapter 4. The higher the result, the better robustness the circuit has. If I_{off}* is higher than I_{on}* the result becomes negative and should be dismissed as failed.

3.2.3 Timing

All digital circuits have to match a certain speed to be able to work together using the same clock. In cells where a clock is not required, it is also advantageous to have high speed, as this will allow a higher clock frequency. Timing in a digital cell is related to the change of the output voltage through:

$$V = V_0 + \frac{1}{C}\int_0^t I_{on} - I_{off}\, dt \qquad \textbf{(3-9)}$$

where V_0 is the initial voltage.

One way of optimizing speed is to find the time a signal needs to change from one logic state to another, and then compare the results. This requires transient analysis and is time consuming.

For a fixed voltage change of 1 Volt, when I_{off} represents a very small contribution, and I_{on} is mainly fixed at its maximum value, this yields a delay of [37]:

$$t_d = \frac{C}{I} \qquad \textbf{(3-10)}$$

As a simplified expression to optimize delay from a DC simulation, we optimized towards the following expression for the delay:

$$t_d* = \frac{C}{I_{on}* - I_{off}*} \qquad \textbf{(3-11)}$$

This is not completely accurate, but works well as a comparison method between cells. The unit is actually second per volt, so it should be scaled by multiplying with the V_{DD}.

3.2.4 Temperatures

For all the calculations and simulations done on the cells, simulations over several temperatures have been done. This is in order to get a more accurate result for cells designed to be used in commercial products. The performance in a CMOS chip is dependent on the temperature the chip is operated in. To get an overall better result the simulations have been done for the temperatures: -20º, 27º, 85º.

3.3 Design parameters

When designing a larger system it is important to decide early on what to design towards and which processes to use. Of the processes we had at our disposal, we chose to use TSMCs 90nm process. The reason for this was that we wanted to be able to synthesize chips from the cell library and send them for prototype production (the 90nm processes is the most common used process at the department and it is affordable). In this Low Voltage/Low Power Standard Cell Library we decided to set the supply voltage to 300mV. This is high in the subthreshold region and will provide

low power and functional robustness.

The performance variables we will be able to change are the widths (W) and lengths (L) of the transistors. We will also be simulating over several temperatures in order to get a better and more reliable result.

3.4 Library content

A standard cell library is a collection of low-level cells. The cells are realized as fixed/double height and variable lengths full custom cells. The most important feature a good library has is to be able to synthesize larger systems with high density and few parasitic effects. If a library only contains a NAND gate, all cells can be synthesized. The design, speed, area utilization and routing will not be so good. Therefore a cell library should contain the most important combinatorial and sequential cells.

3.4.1 Digital Cells

The digital cells composed in this project are:

Combinatorial gates

- NOT (INV/NOT)
  A NOT gate inverts between logic states
- NOT AND (NAND)
  A NAND gate gives a zero if and only if all inputs are high, else the output is high.
- NOT OR (NOR)
  A NOR gate gives a high if and only if all inputs are zero, else the output is zero.
- Exclusive OR (XOR)
  A XOR gate gives a high if only one of the inputs are high, else the signal is zero.

Sequential gates

- Data latch (D-latch)
  The D-latch sets the output to be equal the D input when the input E is zero. If E is high the output is frozen to its last state.
- Delay Flip Flop (DFF)
  The D flip-flop captures the value of the D-input at the falling period of a clock cycle. That captured value becomes the Q output. At other times, the output Q does not change

3.4.2 Cell setup and performance variables

For the NOT, NAND and NOR gates, we use the standard CMOS topology, as seen in Appendix C. The values for the nMOS and pMOS transistors in the circuit were simulated using the procedure that will be described in Section 3.5.

For the NOT gate all lengths and widths were simulated as shown in the

simulation files in Appendix D and explained in Section 3.5.

In equation **(3-12)** it is shown how a single point P in the performance variable matrix is represented, where m, n, o and p refer to a point in the list for its selected performance variable.

To cut down the number of performance variables on the remaining circuits, the NAND and NOR gates were set up to match the NOT gate. This was done by setting the two parallel pMOS transistors in the NAND gate to have the same value as the pMOS transistor found for the NOT gate. The same applies for the NOR gates two parallel nMOS transistors that were set to be the same as the nMOS transistor from the NOT gate. This is illustrated in Table 4-1.

This results in a four dimensional matrix of performance variables for both the NAND gate and the NOR gate. The number of simulations will therefore not be much higher than for the NOT gate. There is still a difference in the number of logic states, these account for approximately 30-50% increase of simulation time (N equal 3 gives 279 values for I_{off}* and I_{on}*). Since the number of performance variables is the same, equation (3-12) still applies, but for the NAND the W_p/L_p is replaced with W_{n2}/L_{n2} and vice versa for the NOR gate. We do however assume that the lengths and widths for the nMOS transistors in the NAND will be equal or not very different. The same applies to the pMOS of the NOR gate.

$$P = \begin{pmatrix} W_p(m) \\ L_p(n) \\ W_n(o) \\ L_n(p) \end{pmatrix} \quad \textbf{(3-12)}$$

$$W_p = \{W_{min}, W_1, W_2, \dots, W_{max}\} \quad \textbf{(3-13)}$$

$$L_p = \{L_{min}, L_1, L_2, \dots, L_{max}\} \quad \textbf{(3-14)}$$

$$W_n = \{W_{min}, W_1, W_2, \dots, W_{max}\} \quad \textbf{(3-15)}$$

$$L_n = \{L_{min}, L_1, L_2, \dots, L_{max}\} \quad \textbf{(3-16)}$$

If we had simulated all 8 performance variables of the NAND and NOR gate, the simulation time would increase exponentially with every extra variable. This would therefore be too time-consuming.

On the DFF there have already been tested low power solutions that should be able to work well [38], but since the simulation of the NOT, NAND and NOR gates was time consuming, it was decided that the rest of the circuits should be built from the already tested circuits. The D-latch and XOR were therefore designed from four NAND gates as shown in Figure C-12 and Figure C-13. The DFF was designed with two D-latches (8 NAND gates) and one NOT gate, as shown in Figure C-14. These circuits will therefore not be the optimal, but they should perform better than the standard cells.

3.5 Optimization strategy and objectives

The simulations are based on netlists created in the schematics tool in Cadence. The devices under test (DUT) have been connected to ideal voltage sources on the input(s) and the output of the circuits. As mentioned in Section 3.3, the constructed digital cells have been decided based on what we believe is a minimum set to synthesize larger designs. The layout of the cells has been done in accordance with the layout rules described in Appendix A. This will be explained in further detail in Section 3.6.

Since the simulation is based on DC simulation, we do not consider the short-circuit power dissipation and the dynamic power dissipation in the simulations, since these need transient analysis to be simulated. From equation (2-4) it is possible to see that the dynamic power dissipation will be reduced by reducing the supply voltage. By measuring I_{off} and I_{on} in the simulated circuits it is possible to calculate the value for static power dissipation, delay and robustness. The optimization algorithm does not need exact values to compare two cases, it only needs a numerical value that sets one case better that the other. This means that we can use simplified formulas.

In order to find more correct values for the I_{off} and $I_{on,}$ Monte Carlo method is used. Monte Carlo simulation and random number generation are explained in [39]. For each set of performance variables, 30 Monte Carlo simulations and one nominal simulation were used over three different temperatures. These temperatures are set according to the working condition the circuits from this library is thought to be used, which is -20º, 27.5º and 85º. This results in 31 sets of I_{off} and I_{on} values, for each of the tree temperatures, for all variations of input values. This will add up to $93*N_i$ values to be computed down to a single value of I_{off} or I_{on}, where N_i is the number of variations of input values.

3.5.1 Voltage sources

When simulating the schematics in the optimization algorithm we will set voltage sources on the input(s) and output of the logic circuit as shown in Figure 3-5. This is done in order to measure the static I_{off} or I_{on} of the circuit. The measurement will as previously mentioned be done at the edges of the noise margin which is 60mV (20% of VDD) for a logic zero and 240mV (80% of VDD) for a logic one.

When optimizing at the edges of the noise margin we will get a very robust circuit. This is because the circuit will be able to operate within specification with poor signals, even though it rarely will encounter them.

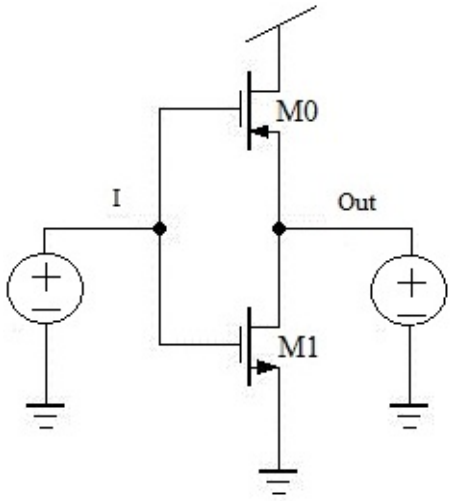


Figure 3-5: Connection of voltage sources on a NOT-gate

For an NOT gate the voltage sources will be:

I	Out
0.2*vdd	0.8*vdd
0.8*vdd	0.2*vdd

Table 3-1: Voltage sources on a NOT-gate

The corresponding tables for NAND and NOR are:

NAND		
I1	I2	Out
0.2*vdd	0.8*vdd	0.8*vdd
0.8*vdd	0.2*vdd	0.8*vdd
0.8*vdd	0.8*vdd	0.2*vdd

Table 3-2: Voltage sources on a NAND-gate

NOR		
I1	I2	Out
0.2*vdd	0.2*vdd	0.8*vdd
0.2*vdd	0.8*vdd	0.2*vdd
0.8*vdd	0.2*vdd	0.2*vdd

Table 3-3: Voltage sources on a NOR-gate

In Table 3-1 we can see that the NOT gate will have 2 variations of input value (N_i will be 2). This will result in 186 values for I_{off} and I_{on}. The NOR and NAND will have three variations for input values (Table 3-2 and Table 3-3), not four as one would expect. This is because the case I1=I2=0 for the NAND will only be a better version of I1≠I2. Since the simulation is a worst case simulation, we save approximately 25% simulation time by ignoring it, without it giving much impact on the results. The same applies to the state I1=I2=1 in the NOR circuit.

3.5.2 Simulation files

The netlists from the circuits are stored as SCS-files and the cadence simulation files are stored as MDL-files. The MDL-file and SCS-file for the NOT gate can be found in Appendix D, the rest can be obtained by contacting the nano-group at Ifi. The SCS-file shows how the values from Table 3-1 are connected in the Netlists for the NOT gate, the MDL-file

shows how the I_{off}* and I_{on}* values are collected.

As shown in the file for the Fitness Function for the NOT gate in Appendix D.4, the calculations for timing, I_{off}* and robustness are carried out with the following calculations:

$$t = \sqrt{mean\left(mean\left(\left(\frac{C}{I_{on}* - I_{off}*}\right)^2\right)\right)} \quad \textbf{(3-17)}$$

$$I_{off}* = mean\left(mean\left(I_{off}*\right)\right) \quad \textbf{(3-18)}$$

$$\text{robustness} = \max\left(\frac{mean\left(log10\left(\frac{I_{off}*}{I_{on}*}\right)\right)}{std\left(log10\left(\frac{I_{off}*}{I_{on}*}\right)\right)}\right) \quad \textbf{(3-19)}$$

These are the same formulas as shown in Section 3.2. The difference is that they are modified so that for each set of performing variables each of these formulas find a single value for timing, I_{off}* and robustness based on $N_i \times 91$ values for I_{off} and I_{on}.

There is one important difference in the formula for robustness. The formula has been inverted to give negative values. This is because the optimization algorithms optimizes towards the lowest value. This means that the lower a number is, the better the robustness is. The closer an answer is to zero the less robust it becomes. And it is these values that are going to be shown and discussed.

For the NOT gate, this gives 186 values for I_{off} and I_{on}, and a total of 693.385 sets of performance variables are simulated. The vast amount of data requires the use of parallel simulations in order to reduce the simulation time. If the simulation of one set of performance variables takes 5 seconds, the total simulation time would be 40 days (statistically the simulations takes closer to 6.2 seconds). This is an inefficient use of resources and the system is therefore set up to handle up to 30 parallel simulations. Simulation sets run independently on separate computer cores. Usually the number of parallel simulations was set to be 25, due to the number of Cadence licenses available and the number of cores allowed to be used simultaneously on the servers. The results from these simulations are only meant to be used as a comparison between different performing variables. The lower the value is, the better is the performance.

After 693.385 simulations we are left with a three dimensional Pareto front with 90.546 Pareto points. These results will be presented in Chapter 4.

3.6 Layout of the library cells

When creating layout for a library cell, there are several elements that have to be thought through. This is the optional settings that we have to decide for ourselves, such as grid size of contacts and routing and height and length of the cells. It is also important to know the most common design rules for the tools and processes used. The most common design rules, library options and calculation on grid size are shown in Appendix A. As seen in the Appendix, the grid size of the library cells is set to 0.33um, the height is 2.64 µm (which is 8 times the grid size) and the width is always an even multiple of the grid size. All the library cells are constructed according to design rules and options explained in Appendix A.

Even though the larger cells are composed of the smaller cells in the library, the layout is still designed to be smaller than a corresponding synthesized cell. As shown in Figure 3-6, all layout designs are confined to using metal layer 1 (M1) and 2 (M2) for routing (the rest of the library cell designs is shown in Appendix E).

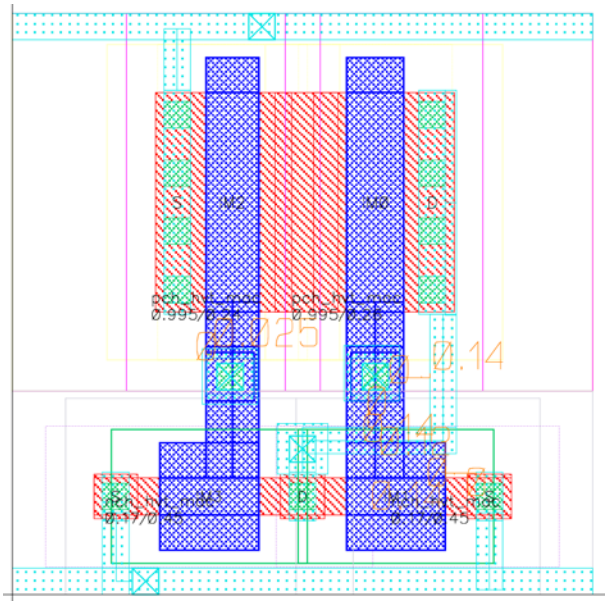


Figure 3-6: Layout of a NOR-gate

After designing the layout, the design will have to go through Design Rule Check (DRC) and Layout Versus Schematic (LVS) Verification. These checks inspect the layout for design errors and look for mismatch between schematic and layout. When there are no important errors (there will be some density errors, but since this is only a part of a larger design, they will be ignored in this step), Parasitic Extraction (PEX) will calculate the parasitic components and generate a PEX-file for simulation purposes. The simulation of the schematic only takes into account transistor effects. The PEX generator generates theoretical parasitic capacitance and parasitic resistance that is caused by the resistance and potential voltage difference in and between wiring, doped substrate and interconnections (the PEX tool can also be set to calculate parasitic inductance). There are 26 parasitic resistors and 38 parasitic capacitors only in the NOT gate.

In order to be used in a library, these cells must also go through

different generators for extraction and creation of data for the synthesize tool. More information about this can be found in [40] (which is an unpublished tutorial that can be obtained from the nano-group at Ifi).

3.7 Synthesis of a 32-bit adder

In order to synthesize a design, the library needs to be extracted. An extracted library is a list of cells, with nets, parasitic components and wires.

When synthesizing a circuit from a standard library, there are several steps that need to be done. After deciding what circuit that is to be synthesized, we must find a suitable way to describe the circuit. For this we used a hardware description language called VHDL. This is a programming language used to describe circuits. When the VHDL is written, it can be converted into Verilog HDL code by a converter. This converter sets up the Verilog code with the library cells from the new library and acts like a direct roadmap for the signals and cells.

3.7.1 VHDL and Verilog

The design we decided to synthesize was a 32-bit signed adder with carry. The VHDL code for this is shown the box below and comes from [41].

```
library IEEE;
use IEEE.std_logic_1164.all, IEEE.numeric_std.all;
entity ADD32 is
        generic (n: NATURAL := 32);
        port ( A, B : in std_logic_vector(n-1 downto 0);
                Cin : in std_logic;
                Sum : out std_logic_vector(n-1 downto 0);
                Cout : out std_logic);
end entity ADD32;

architecture sgned of ADD32 is
        signal result : signed(n downto 0);
        signal carry : signed(n downto 0);
        constant zeros : signed(n-1 downto 0) := (others => '0');
begin
        carry <= (zeros & Cin);
        result <= (A(n-1) & signed(A)) + (B(n-1) & signed(B)) +
carry;
        Sum <= std_logic_vector(result(n-1 downto 0));
        Cout <= result(n);
end architecture sgned;
```

VHDL of a 32-Bit signed adder

The code was then sent through a converter together with library and cell information. This was an important step because we can with the Verilog code see if the converter tool (Encounter RTL Compiler) understands the logic functions of the library cells (the Verilog code is

partially shown in the box below).

```
module add_signed_carry(A, B, CI, Z);
 input [31:0] A, B;
 input CI;
 output [32:0] Z;
 wire [31:0] A, B;
 wire CI;
 wire [32:0] Z;
 wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
 wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
-----
 NANDX1 g2244(.I1 (n_131), .I2 (n_320), .Q (n_322));
 NORX1 g2248(.I1 (n_58), .I2 (n_318), .Q (n_320));
 INVX1 g2690(.A (n_61), .Q (n_60));
-----
 NANDX1 g2691(.I1 (B[7]), .I2 (A[7]), .Q (n_61));
 XORX1 g2765(.I1 (n_175), .I2 (n_292), .Q (n_404));
-----
endmodule
```

Excerpt from the Verilog code

The Verilog code lists all pins, wire names and cell names with instance number that will be used in the design. If a signal is followed, one will find out which other signals and components it comes in contact with. In the Verilog code above we can see that the signals A<7> and B<7> enter the inputs on a NAND gate with instance name g2691. The output is connected to a wire called n_61. If we were to follow the path of n_61, we would see that it would be connected to the carry network and be used to decide the output of Z[7] together with the carry network.

3.7.2 Verification of the synthesized adder

After the VHDL code has been converted, the Verilog file can be imported to virtuoso and simulated. This is usually done in order to verify the design and the logic function of the entire design. The Schematic can be found in Appendix G.1.

There are two ways this schematic can be simulated: with or without parasitic components. Either way we have to create a test bench order to test the adder. This is done by adding together the number "10101010.....10" (which is positive in a signed circuit) with the number "01010101.....01" (which is negative). This becomes "11111111.....111", which is equal to -1. A carry in (Ci) signal is then applied to the input, making the answer 0, which is all 33 (32 + carry) outputs equal to zero. To do this, the circuit needs to ripple through all the 32 bits to calculate and present the output.

3.7.3 Design synthesis and place and route

When the Verilog code is confirmed operational, the next step is to synthesize the design. This is done in encounter. Here we import the

Verilog code from the previous Section with the LEF-files extracted from the library. The LEF files tell us the physical properties of the different layers of the cells and are used to set boundaries for routing in the synthesizer. The LEF files also contain technology information. Technology information is information about the VIAs, wiring, pins and metal layers. The synthesizer can synthesize chips with quite good density, but this requires a lot of computing power. If the chip density is set to 60-70%, the computations can be done fairly quickly with a normal eight-core computer. The synthesizer then places the library cells, filler cells, power rings and pins. We can manually go in and adjust settings in some of these steps if we want. Then we run the routing tool and the synthesizer is finished.

After this step we have to export the design to a DEF file, which is a Design Exchange Format, and import the file in virtuoso. Here we can run DRC and LVS and fix small errors in the design. Then the parasitic components can be extracted and the design is ready for simulation.

The finished design will now go through a series of function tests and simulations. This is done in the same manner as the verification of the Verilog code in Section 3.7.2 and shown in Section 4.3.

Chapter 4

4 Results

From simulation and optimization a small standard cell library has been set up. This means that the circuits explained in detail, the NOT, NAND and NOR, have been completely designed with all the needed specifications to work in a library. They have also been tested to work together in a larger non clocked design.

In addition to this the XOR gate and filler cells were successfully designed and function tested. The XOR that consists only of four NAND gates was not crucial since the synthesizer could design this cell itself from NAND or NOR gates, but in order to make the design more compact, it was designed, and then the variant with NAND gates was chosen, since the NAND gate was a closer match to the NOT gate.

The D-LATCH and DFF were also created in schematic, but ran into some small problems when making the library definition. The synthesizer must therefore at the moment synthesize these circuits itself. Since these circuits can be made of NAND and NOT gates, like I have done, or with NOR gates, the synthesizer will be able to realize designs that require these types of logic.

4.1 Optimization results

The optimization process was one of the most time consuming sections of this project. The optimization results presented here are the result of several runs with different set-ups, where the presented sets were the sets considered most accurate. As mentioned in Section 3.4.2, the pMOS transistors of the NAND and the nMOS transistors of the NOR were set equal to the pMOS/nMOS of the NOT gate. This was done to cut down the simulation time drastically.

The values for the three simulated logic cells are as follows:

Name	**NOT**	**NAND**	**NOR**
pMOS(M0) (W/L)	815n / 400n	815n / 400n	995n / 240n
pMOS(M2) (W/L)	-	815n / 400n	995n / 260n
nMOS(M1) (W/L)	450n / 170n	340n / 200n	450n / 170n
nMOS(M3) (W/L)	-	340n / 200n	450n / 170n
Timing (s/V)	4.322e-05	5.767e-05	5.094e-05
I_{off}* (pA)	4.509e-11	5.147e-11	1.078e-10
Robustness(-μ/σ)	-6.324	-6.218	-4.217

Table 4-1: Simulation results of optimized cells.

These values are found as explained in Section 3.5.2. M1 to M3 is the name of the transistors in the schematic and can be seen in Appendix C. The calculations have included the values from simulation over all the three temperatures previously mentioned. These values are the values used in the layout of the design. The reason we chose these specific values is described below. All the larger cells are based on these cells and are therefore a product of the simulations.

4.1.1 NOT gate

After simulating the NOT-gate, we are left with a large matrix of results. As mentioned in Section 3.5.2, there are 90.546 sets of points in the Pareto Front for the NOT-gate. Each set contains a value for delay, robustness and static power dissipation that corresponds to a set of L/W coordinates. The values of the Pareto points are presented in the three graphs below. From this matrix we must extract a single result to use.

The first step of finding a single value was to clear out all results that were not robust enough, then we concentrated on the I_{off}* and timing. This means that the final set of coordinates was mainly chosen based on the values in Figure 4-1. From these graphs three results were chosen, one for high speed (HS), one for low power (LP) and one in the middle (NOM). The one in the middle is the one we chose to continue to work on and base the other circuits on. The sizes and results can be found in Table 4-2.

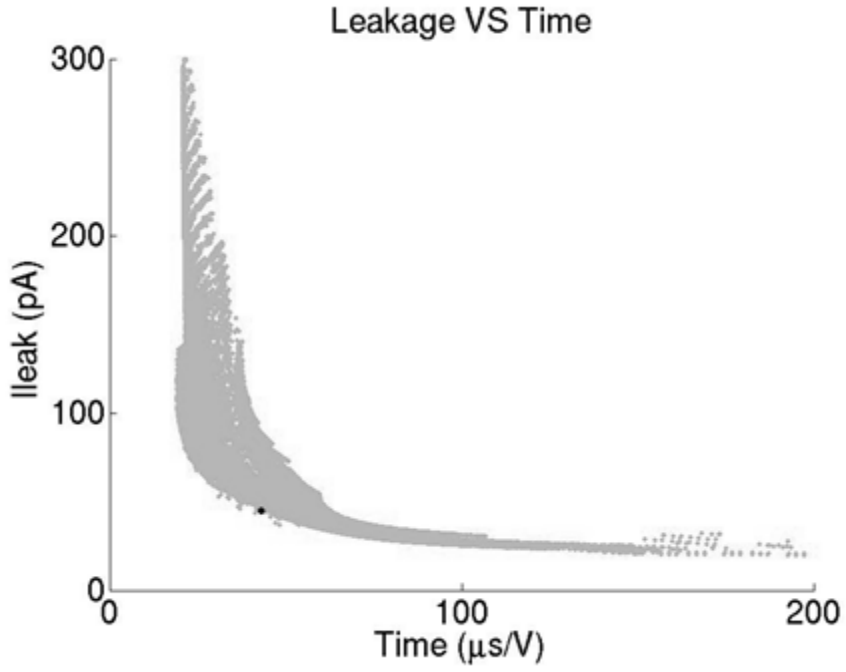


Figure 4-1: Pareto front leakage-time for the NOT gate

Figure 4-2 shows the Pareto Front of time vs. robustness. From this plot the effects of simulating 3 temperatures and several variables can be observed as distinct patterns.

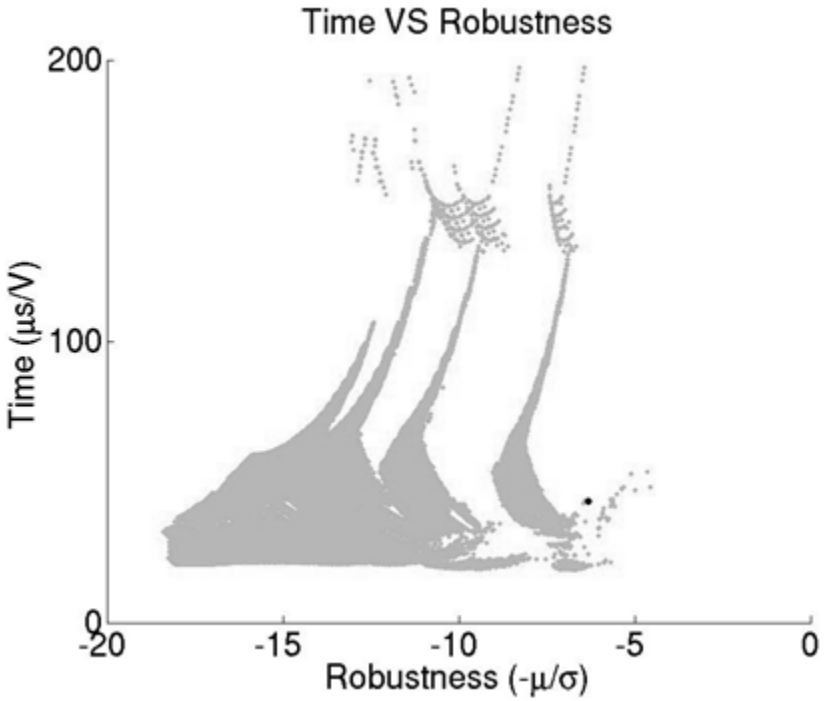


Figure 4-2: Pareto front time-robustness for the NOT gate

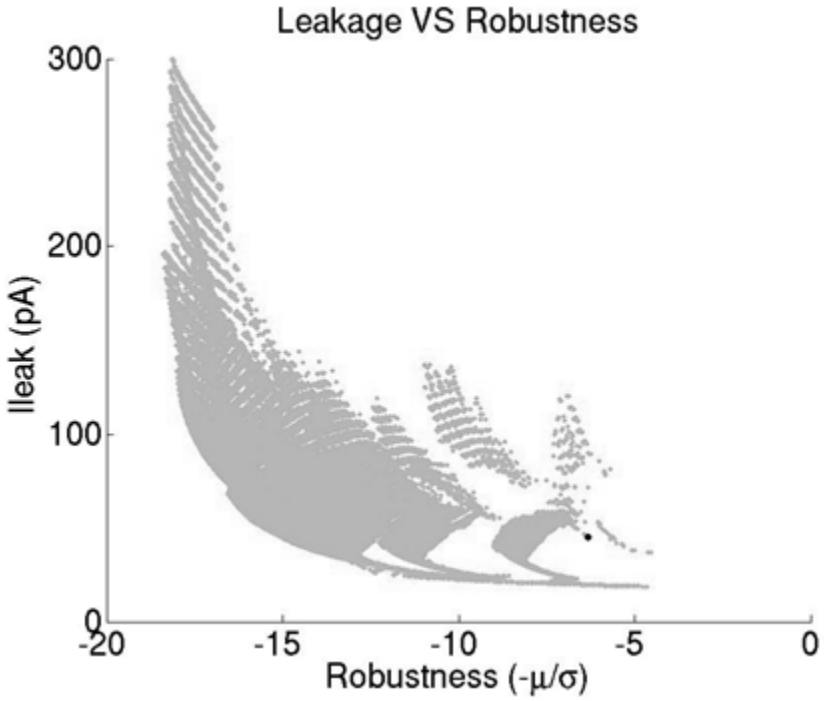


Figure 4-3: Pareto front leakage-robustness for the NOT gate

The chosen result is marked with a black circle in the plots (Figure 4-1 to Figure 4-3).

From this we get the result relisted below.

	NOT (HS)	NOT (NOM)	NOT (LP)
pMOS (W/L)	1455n / 240n	815n / 400n	120n / 970n
nMOS (W/L)	130n / 560n	450n / 170n	130n / 3360n
Timing	20.9 µs/V	43.2 µs/V	131.8 µs/V
I_{off}*	86 pA	45 pA	24 pA
Robustness(-µ/σ)	-6.0124	-6.324	-7.0124

Table 4-2: NOT gate sizes and values

This is a circuit that is easy to realize and the layout is shown in Appendix E.

4.1.2 NAND gate

As explained in Section 3.4.2, both pMOS transistors, instance M0 and M2 (shown in Figure C-10 in Appendix C.2), were set to be the value of the pMOS transistor from the NOT gate. The inputs are also as mentioned set to the values seen in Table 3-2. This was to match the pull-up network in the NAND with the NOT-gate. After the simulation of the NAND gate, we thought that there were some missing points in the simulations. Therefore a new simulation was started over a smaller area. This resulted of course in fewer results in the finished simulation. This is why the plots below have fewer results and can seem more random.

The solution was chosen in the same way as with the NOT gate. The final values were based on the best correspondence between timing and I_{off}* after clearing the points with a robustness lower than acceptable.

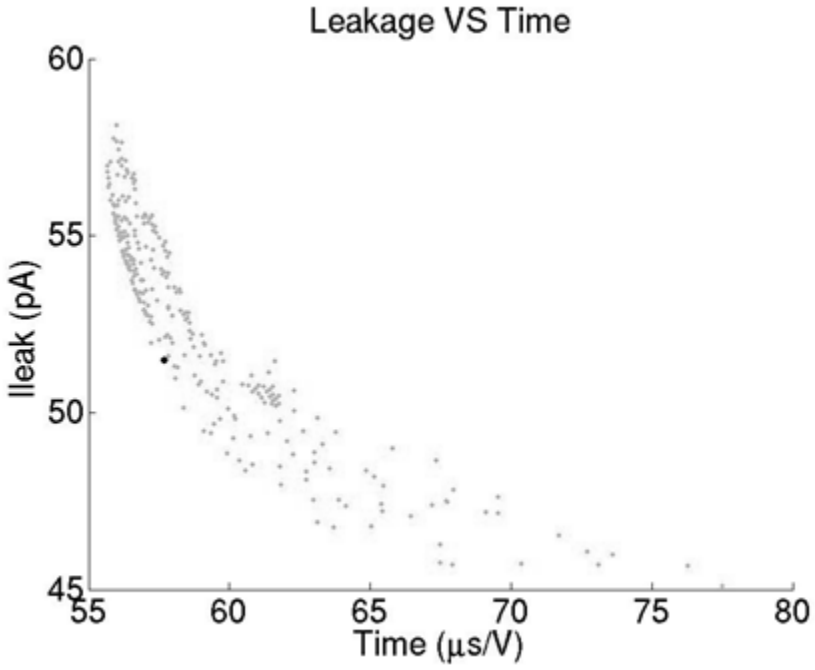


Figure 4-4: Pareto front leakage-time for the NAND gate

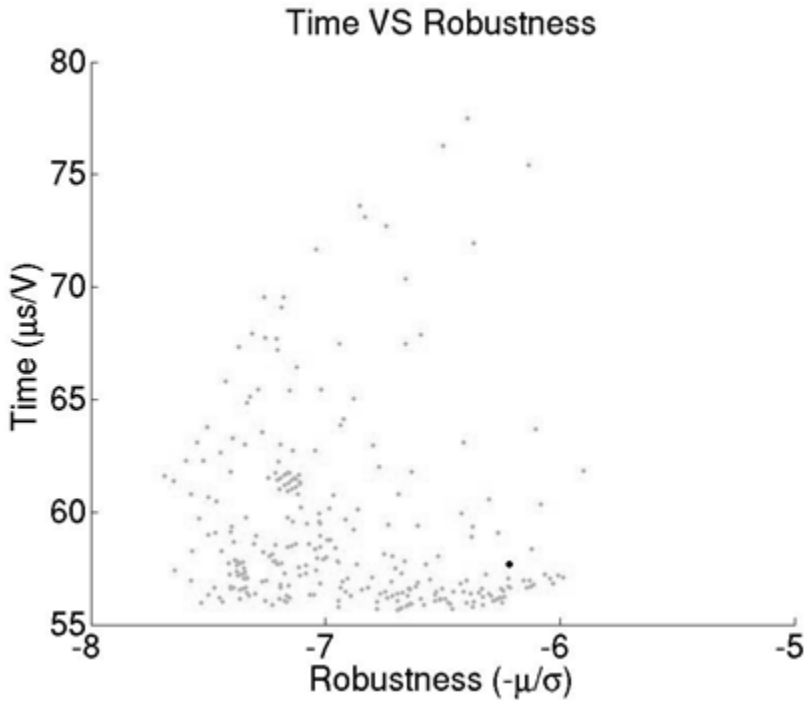


Figure 4-5: Pareto front time-robustness for the NAND gate

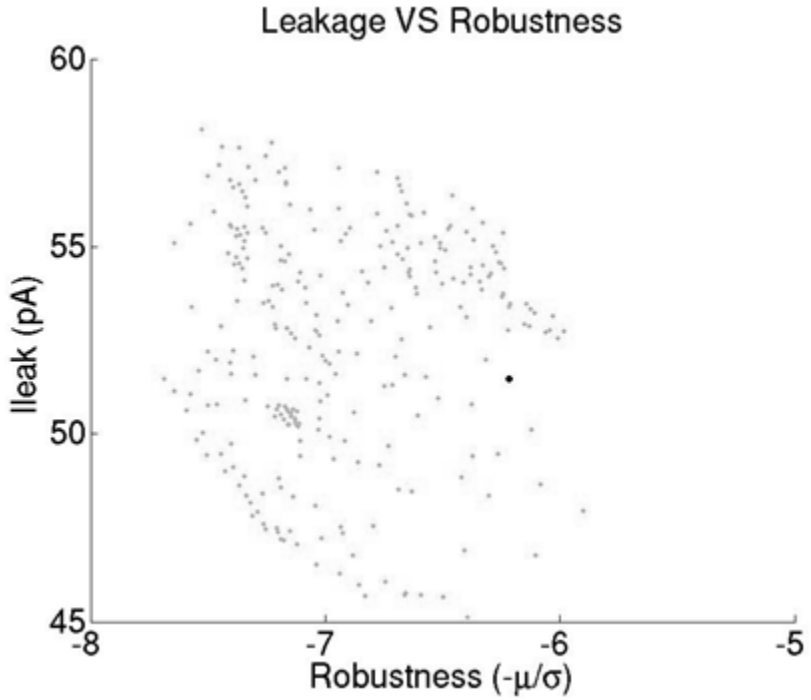


Figure 4-6: Pareto front leakage-robustness for the NAND gate

The results are shown as the black circles on the plots (Figure 4-4 to Figure 4-6) and are listed with input values (length and widths) and output values (delay, robustness and I_{leak}) in Table 4-3.

pMOS(M0) (W/L)	pMOS NOT
pMOS(M2) (W/L)	pMOS NOT
nMOS(M1) (W/L)	340n / 200n
nMOS(M3) (W/L)	340n / 200n
Timing	58 µs/V
I_{off}*	51 pA
Robustness	-6.218

Table 4-3: NAND gate sizes and values

The timing and I_{off}* is only slightly higher than for the NOT gate. Both are about 20% higher in value. The robustness is about the same.

4.1.3 NOR gate

The simulation process for the NOR was the same as for the NAND. Even the fitness function was the same (because of the same number of input values and same response from the MDL-files). Here the nMOS transistors M1 and M3 (from Figure C-11 in Appendix C.3) was set equal to the nMOS of the NOT gate (values in Table 4-2) and the input values, I_1 and I_2, were set to be the values from Table 3-3. This gives us a Pareto front that looks like:

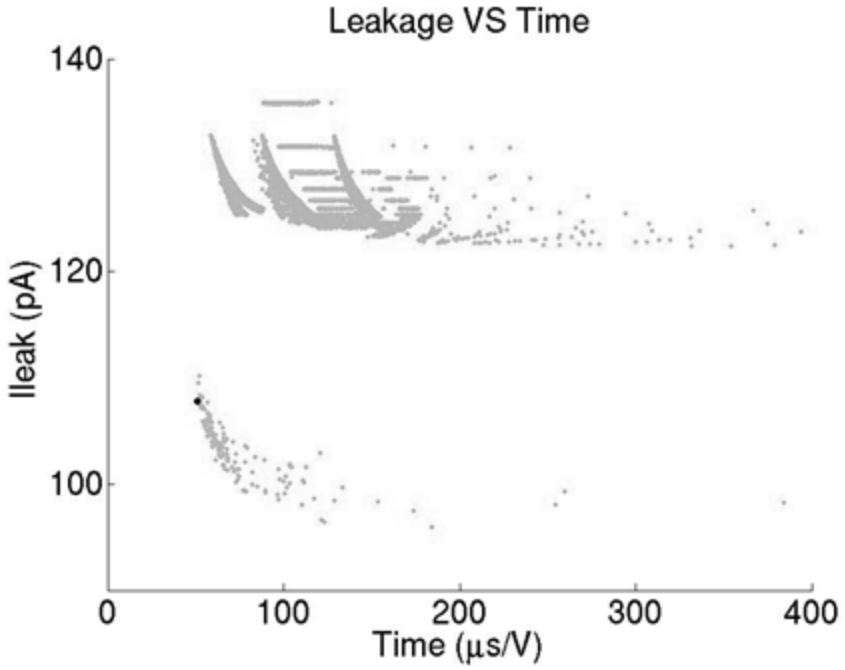


Figure 4-7: Pareto front leakage-time for the NOR gate

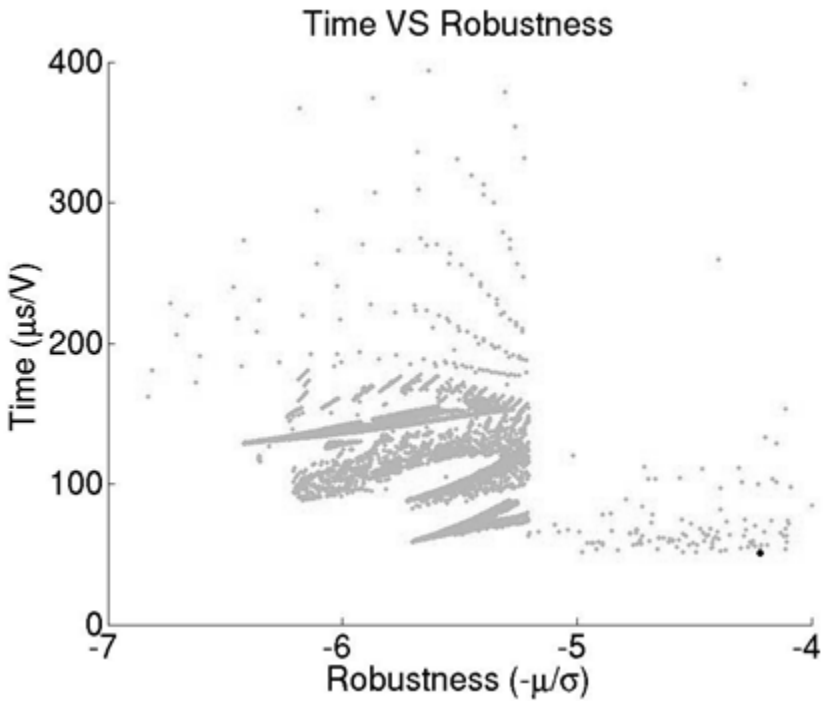


Figure 4-8: Pareto front time-robustness for the NOR gate

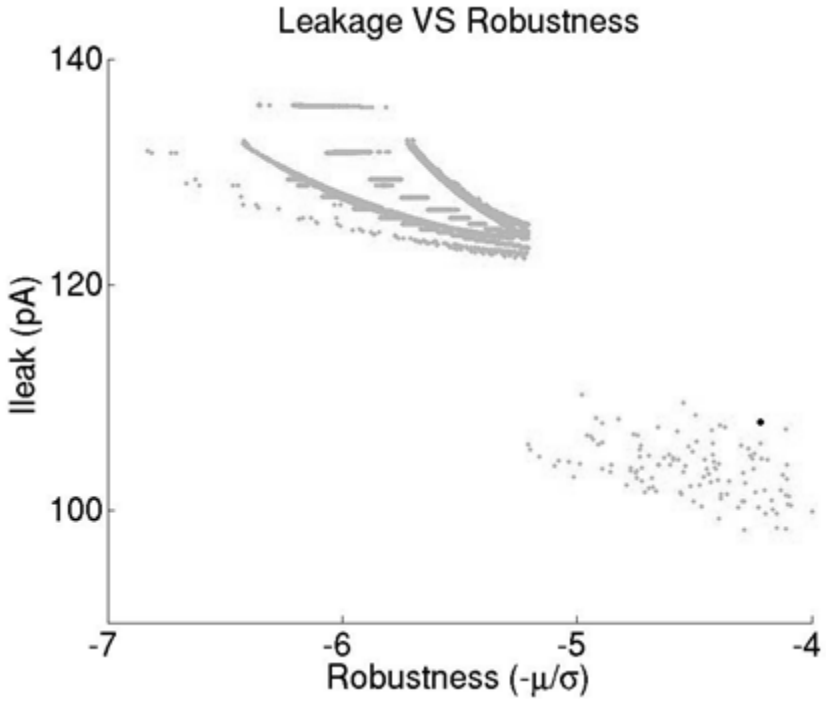


Figure 4-9: Pareto front leakage-robustness for the NOR gate

In Figure 4-7 to Figure 4-9 we can see, as in the previous cells, that a point with a good delay and I_{off}* is not among the most robust points. For the chosen values, the delay of the NOR gate is about the same as the delay for the NAND gate. The I_{off}* on the other hand is about the double.

pMOS(M0) (W/L)	995n / 240n
pMOS(M2) (W/L)	995n / 260n
nMOS(M1) (W/L)	nMOS NOT
nMOS(M3) (W/L)	nMOS NOT
Timing	51 µs/V
I_{off}*	108 pA
Robustness	-4.217

Table 4-4: NOR gate sizes and values

4.2 Results from library cells

After the layouts of the cells described in Section 4.1 have been designed and the parasitic components have been extracted with the PEX tool, the cells can be simulated again. In Appendix F.1 the behavior of the NOT gate with and without parasitic components is shown as graphs and the information about the delay, energy and effect is listed in Table F-7 in the Appendix.

4.2.1 Delay and power dissipation of a NOT gate

To find the delay and power dissipation of the NOT gate we set up a simple simulation. To find the delay, a chain of three NOT gates where set up and the delay was measured between the second wire (t1) and the third wire (t2), as shown in Figure 4-10. The reason for the first NOT gate was to even out the signal from the signal source to match the rise and fall time of the circuit.

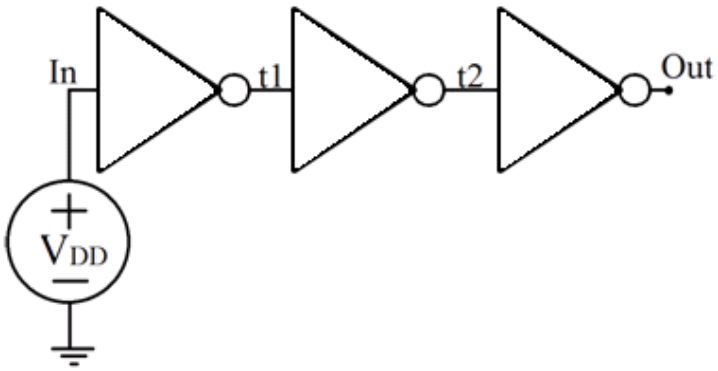


Figure 4-10: A chain of NOT gates

The power dissipation is measured with the current in the supply voltage source when the NOT gate is connected in loop as shown in Figure 4-11. The current used is the one that the circuit settles down on after 5µs. This is because we want the static current. The wires in Figure 4-11 are given separate starting values to avoid getting them stuck in VDD/2 (one equal VDD, the other equal ground).

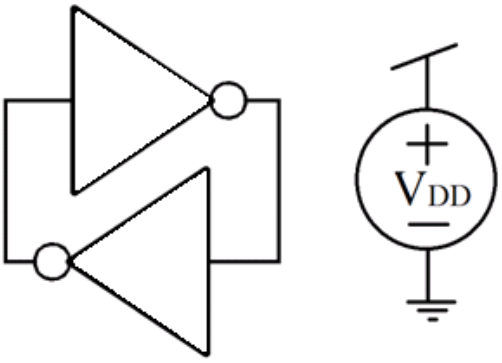


Figure 4-11: A loop of NOT gates

On both simulations we used Monte Carlo method with 5000 runs on three different temperatures (-20º, 27º and 85º). This gave us the range of values presented in the histograms below.

From the results we can clearly see that the static power dissipation is going up dramatically in accordance with the temperatures, but at the same time the delay goes down. When the temperature moves from -20 to 27 the mean I_{leak} goes up with more than one decade ad it goes up another decade from 27 to 85, this can be seen in Figure 4-12 to Figure 4-14. In the same way the mean delay reduces with a decade for the same temperature movement, as shown in Figure 4-15 to Figure 4-17.

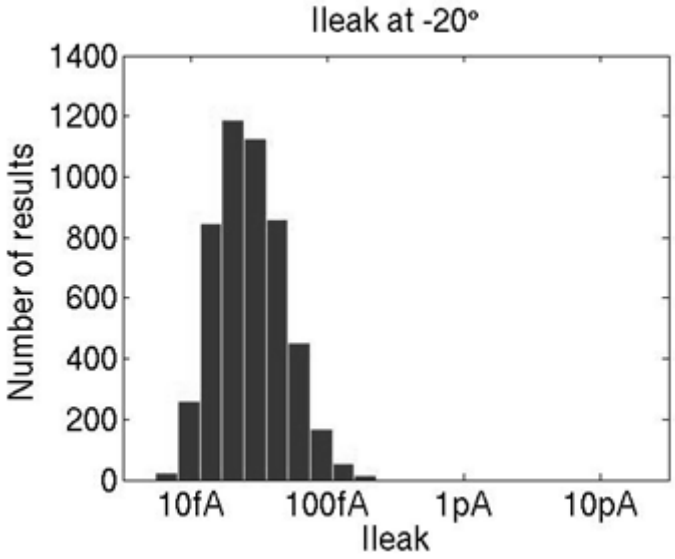


Figure 4-12: I_{leak} at -20° in the loop of NOT gates

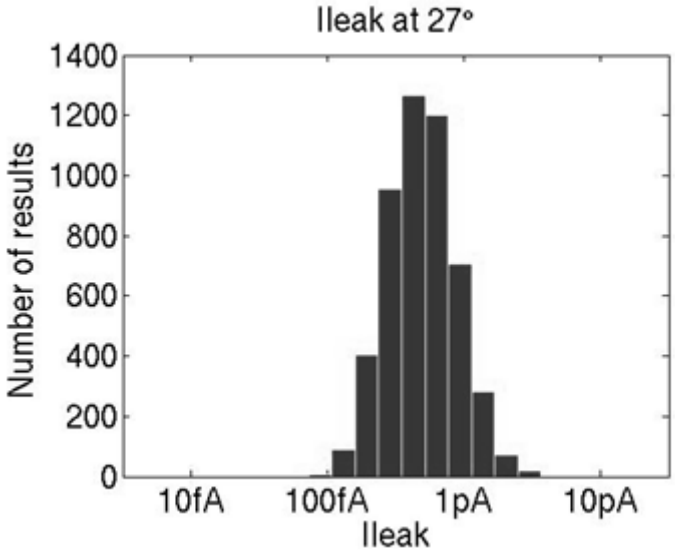


Figure 4-13: I_{leak} at 27° in the loop of NOT gates

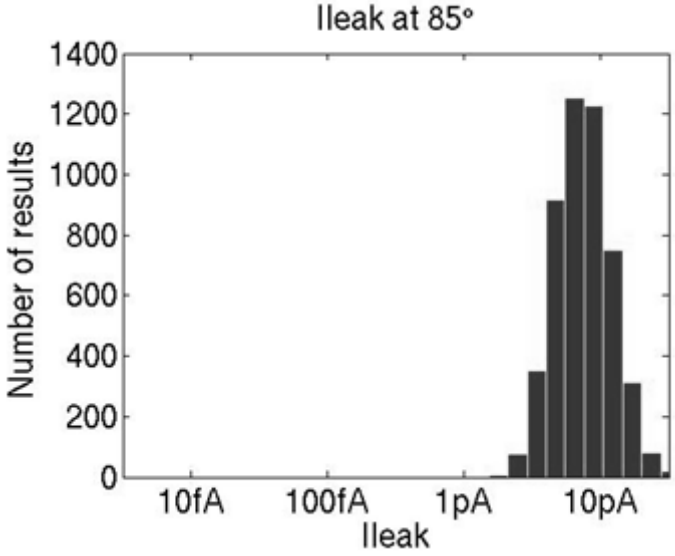


Figure 4-14: I_{leak} at 85° in the loop of NOT gates

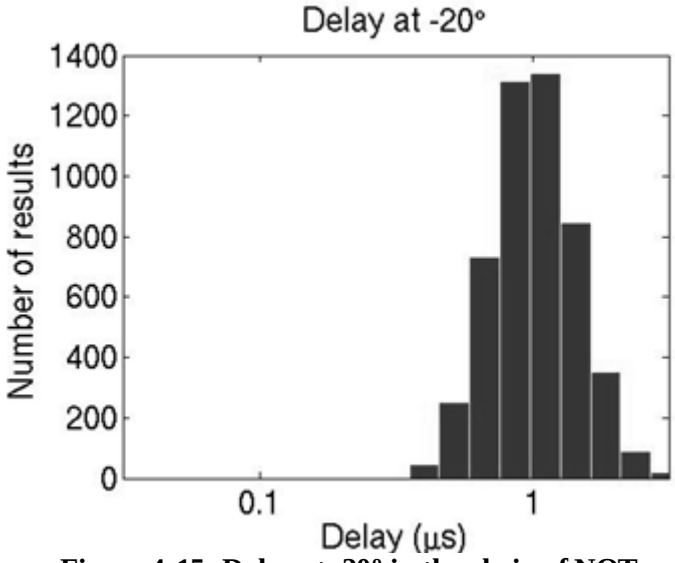


Figure 4-15: Delay at -20° in the chain of NOT gates

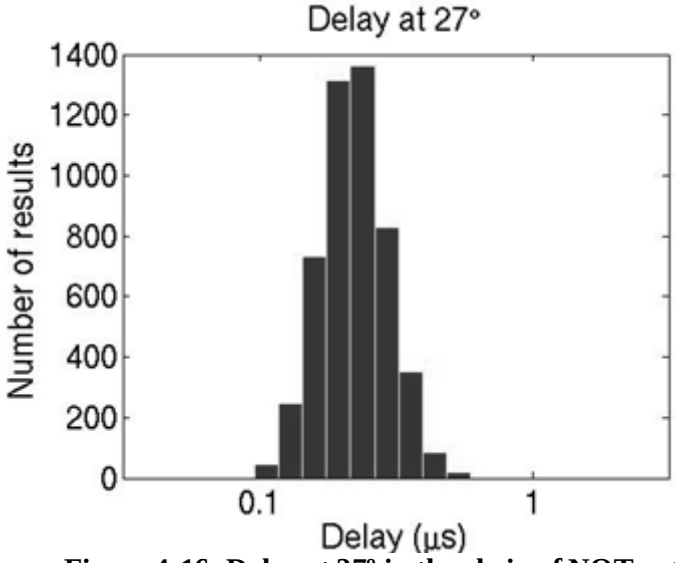


Figure 4-16: Delay at 27° in the chain of NOT gates

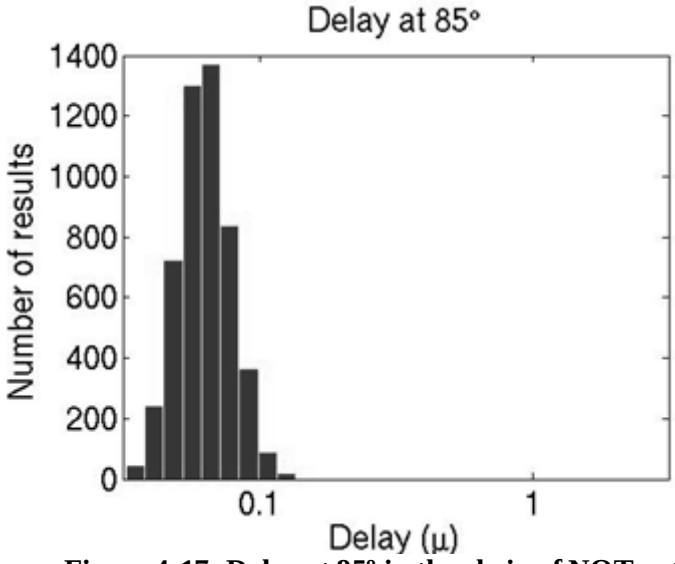


Figure 4-17: Delay at 85° in the chain of NOT gates

4.2.2 Matching of cells

To show how the NAND and NOR gate are matched to the NOT gate, two simple tests are carried out. The NAND and NOR gate are simulated as NOT gates in the same way as in Figure 4-10 and Figure 4-11. This is done by connecting one of the inputs to the supply source when simulating the NAND and setting one input to ground when simulating the NOR (this makes the NAND and NOR gates into NOT gates). It would also work by setting I_1=I_2 for both cases, but that would give us the best case. From these simulations we get the following tables:

	-20º		**27 º**		**85 º**	
NOT	**0.14 pA**	0.17 pA	**3.1 pA**	3.2 pA	**48.8 pA**	48.8 pA
NAND	**2.3 pA**	2.3 pA	**5.3 pA**	5.6 pA	**53 pA**	57.8 pA
NOR	**2.4 pA**	2.4 pA	**9.3 pA**	10 pA	**119 pA**	126 pA

Table 4-5: Static current through supply source with and without (bold) parasitic components

	-20º		**27 º**		**85 º**	
NOT	**773 ns**	996 ns	**171 ns**	218 ns	**50 ns**	61 ns
NAND	**1089 ns**	1379 ns	**244 ns**	305 ns	**71 ns**	87 ns
NOT	**1165 ns**	1427 ns	**209 ns**	249 ns	**49 ns**	57 ns

Table 4-6: Delay with and without (bold) parasitic components

In the simulations without parasitic components, we see that the delay (Table 4-6) and the static current (Table 4-5) in the different temperatures are in the same order of magnitude. Some deviation is expected.

In the simulations with parasitic components, all the results from the circuits increase about the same amount.

If the I_{off}* of the NOR is about the double the I_{off}* of the NAND, the delay for the NOR should be almost equal to the delay of the NAND. This is shown in Table 4-1.

This corresponds with the simulation values in the tables above.

A Monte Carlo simulation on this would probably shed more light on the actual difference.

4.3 Synthesized design

The circuit synthesized in this thesis is a 32 bit signed adder with carry. The layout shown in Figure 4-18 has a large area and consists of 67 NOT, 129NOR, 36 NAND and 31 XOR gates (where each XOR consists of four NAND gates). In addition, there are 67 filler cells with "dummy" structure and the rest is fillers with metal wiring and well area. There are 220 well contacts spread over the design. The entire design is 86.78µm×78.31µm with power rings and contacts, the layout measures 66µm×58.08µm which is 200×176 grid sizes.

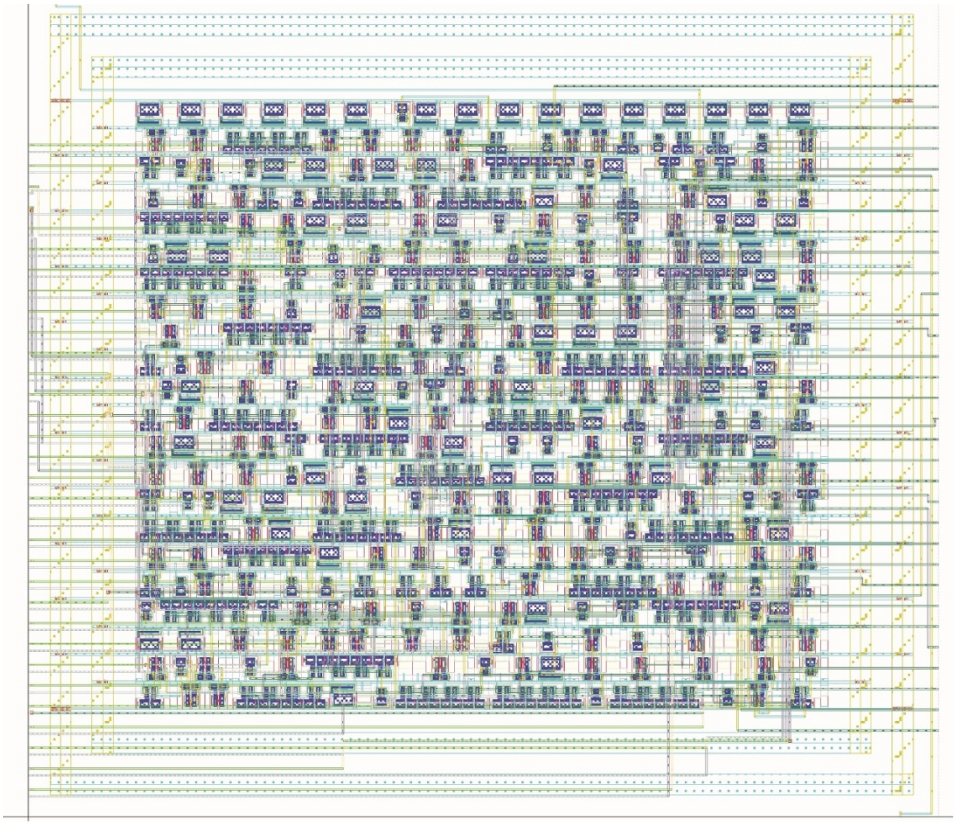


Figure 4-18: Layout of a 32-bit signed adder with carry

This circuit contains many parasitic components and is not the most optimal circuit it is possible to synthesize. There are several options in the synthesize tools that have not been explored.

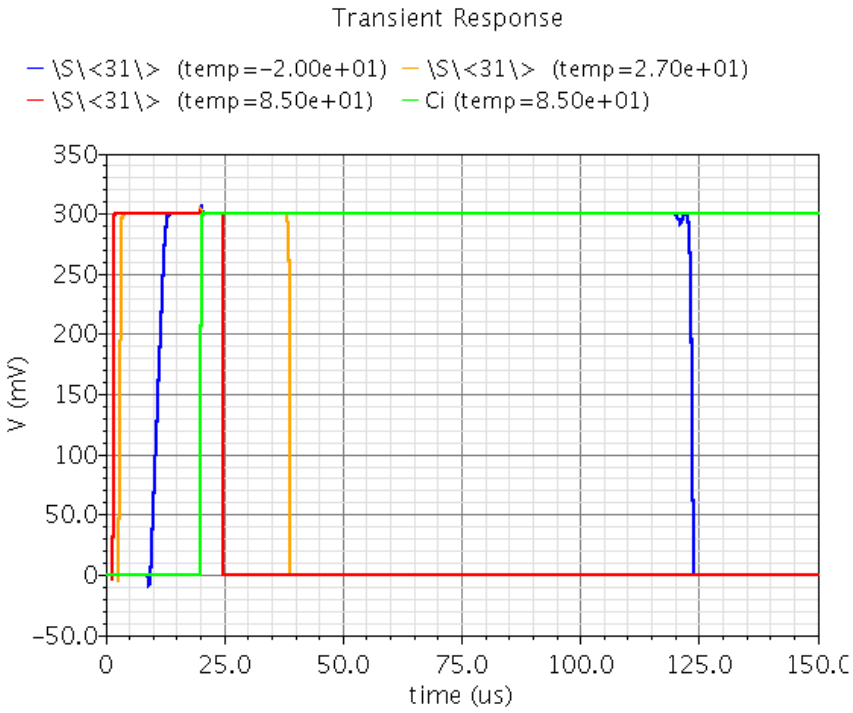
4.3.1 Adder simulation with parasitic components

When simulating this circuit with parasitic components, it is set up as explained in Section 3.7.2. On the adder the inputs A and B give signal after 1 µs. The output response is shown in Graph 4-1. After 20 µs the carry-in (Ci) gives a high input. At this point all the outputs should go low. The blue, orange and red graphs in Graph 4-1 are the output of the 32-bit adder at respectively -20º, 27º and 85º.

In Table 4-7 the delays are shown. The "best" case is when all the outputs flip at the same time and the carry network is not used (A=101010… and B=010101…). The case with ripple is when the carry sends a signal in and has to ripple through the entire design.

In Appendix F.2 the results of the following simulations can be found:

- Simulation without parasitic components (schematic).
- Simulation with parasitic components on cell level (Verilog code).
- Simulation with parasitic components on design level (synthesized layout).



Graph 4-1: Output response of a 32-bit adder

Temp	Best	Ripple
-20.00	11.23 µs	103.5 µs
27.00	3.08 µs	18.68 µs
85.00	1.60 µs	4.646 µs

Table 4-7: Delay in the 32-bit adder at nominal simulation

In order to see the difference between a supply voltage of 300mV and 1.2V the following test was done:

The circuit was set up with a supply voltage of 1.2V and simulated as the 300mV version above. This is the same circuit, with the same parasitic components. The result is presented in Table 4-8

Temp	Ripple 1	Energy 1	Ripple2	Energy 2
-20.00	103.5 µs	66 fJ	25.66 ns	1.5 pJ
27.00	18.68 µs	60 fJ	27.97ns	1.6 pJ
85.00	4.646 µs	68fJ	30.97ns	1.7 pJ

Table 4-8: Comparison between high and low supply voltage on the 32-bit adder

Ripple 1 is the delay from Table 4-7 and is from simulations with a supply voltage of 300mV and parasitic components on design level. Energy 1 is taken from Graph F-4 in Appendix F.2 where the supply voltage is 300mV and there is simulated with parasitic components on design level. The energy showed is the consumed energy for one calculation in the 32-bit adder.

Ripple 2 is the same simulation as Ripple 1, but with a supply voltage of 1.2V. Energy 2 for Ripple 2 the same as Energy 1 is for Ripple 1.

Chapter 5

5 Discussion

5.1 Optimizing algorithms

A multiobjective parameter search using the grid based algorithm (Appendix D.3) with Pareto optimality was used in the search to find a fitting Pareto front for timing, static leakage and robustness. The reasons why we used this optimization method instead of the *gamultiobj* optimization, were in order to save simulation time and get values that were on grid. The tools that are used to build the circuits have a minimum grid size that we have to relate to. The *gamultiobj* algorithm might use a great deal of time to simulate in the area around a valid point, without simulating the point.

The timing of the cells was found as the RMS value of the Monte Carlo simulations, the static leakage as the mean value and the robustness as the worst case. These are the values presented in the plots in Section 4.1. In order to reduce the simulation time, several choices for the circuits were made in advance. This was in order to reduce the number of performing variables on the NAND and NOR gate and limit the simulation span of the performance variables.

Optimizing objectives for better circuits is of the highest importance for producing modern standard cells for libraries and cells for VLSI circuits. The optimizing methods discussed in this thesis and the optimization algorithms and fitness functions used, are quite versatile and can be used on most optimization cases. The multiobjective optimization with Pareto efficiency is an optimization method that successfully can be applied to all cases that require optimization of more than one objective.

The fitness function and optimization algorithms used in this thesis have been customized to be used directly on the different cells. It would not be very difficult to rewrite them to be used for several different cases, and not only for similar cases. With some modification the fitness function and the grid based multiobjective optimization algorithm should be able to work universally on circuits with different number of performance variables and outputs.

The fitness function can also be modified not only to run parallel on the same circuit (with different performance variables) but also to also run several parallel Monte Carlo sets. This can enhance the simulation time of Monte Carlo simulations and increase the number of circuits simulated at a given timeframe.

The fitness function that was customized to work on the simulated cells, the NOT, NAND and NOR gate, shows how effective the optimization work can be done when multi threading is used.

5.2 Reliability requirements

When we have simulated and optimized the library we have set several constrains and taken some choices concerning transistor type and simulation temperatures. The transistors were sett to be high threshold voltage transistors (450mV). This was in order to decrease the power dissipation of the circuit, which also increases the robustness. A higher threshold voltage has the effect that the timing becomes worse. The temperatures were set to be -20º, 27º and 85º. These temperatures are within the scope of the working conditions.

The results from the simulations are listed in Table 4-1 and have been chosen from the plots presented in Chapter 4. Because we simulated over three different temperatures, on the edges of the noise margin, and only took out respectively the worst case, the mean case or the RMS value, the circuits should in almost all cases work much better than presented.

The simulated NOT gate presented in Section 4.2.1 has a delay of 20.9µs/V, which is 6.27µs, and the I_{off}* is simulated to be 45pA. When simulating the layout of the NOT gate with parasitic components on 300mV, the results are, as assumed, better than from the initial simulations. In Figure 4-12 to Figure 4-14 we see that the static leakage (I_{off}) ranges between 10fA to 20pA depending on the temperatures. This is better than 45pA. In Figure 4-15 to Figure 4-17 the timing is shown to rage between 50ns to 3µs depending on temperature.

Due to the simulation setup specifications, symmetrical behavior on the circuits could not be achieved and was not a criteria in the optimization process. As we can see in Graph F-2, the NOT gate does not have matched rise-fall time. The largest difference can be seen for -20º, were the rise time of the NOT gate is much larger than the fall time. A slow rise-fall time usually gives higher dynamic power consumption, but as it can be seen from Table F-7, it does not result in much fluctuation in the dynamic power consumption across the specified temperature range.

When clearing out values that should not be chosen, we set the limit of acceptable robustness to be -3 (μ/σ). Since the robustness simulations were done at the edges of the noise margin, and only the worst case of the temperatures was chosen, the robustness should, such as timing and static leakage, be much better than presented in the Pareto front.

5.3 The optimized cell library

A properly designed cell library is essential for synthesizing larger digital ASIC designs. The library has been created on the background of the simulations shown in Chapter 4. The schematic and truth tables for the cells can be found in Appendix C, and in Appendix E the layout is shown. This library is more or less functional. All the cells are designed according to design guidelines and the files that are needed in order for it to synthesize larger designs, have been created.

In the area of interest, there were several close points. As explained in Section 3.1.1, some sizes are easier than others to realize in an actual circuit. The last deciding factor was therefore the input lengths and widths for the results. This was in order to design a library cell that fits the sizing

criteria, which was set to be eight times the grid size in height. It was also done in order to have short routing and compact cells, which will reduce parasitic components.

The three main cells that were designed, work according to our specifications, but there is still much that can be done with the library. It might be useful to increase the simulation range to see if larger cells can work better, or to simulate on all performance variables (free range simulation on all n- and pMOS transistors in the cells). This was done for the NOT gate, but not for the NAND and NOR.

One drawback was that the DFF and D-LATCH were not completed. This was because there were detected problems in some of the last simulations before synthesizing the 32-bit adder. We decided not to use too many resources on this since the synthesizer would be able to synthesize these cells itself. The design of these two cells was the simplest and not the fastest or most energy efficient. Other topologies should therefore be tested.

5.4 The synthesized adder

As a concept design, we chose a signed 32-bit adder with carry. It was chosen because it is a fairly simple circuit that could be built from the four completed library cells, the NOT, NAND, NOR and XOR, and yet complex and large enough to give the synthesizer some work and test the library. The VHDL code was also easy for the converter to convert to a readable Verilog code that is easy to debug.

The synthesized circuit was simulated on schematic level (no parasitic components), cell level (parasitic components from the cells, but not from full design) and on design level (parasitic components for the entire layout). This was done in order to see how the parasitic components affect the circuit.

In Appendix F.2 the simulation graphs from the 32-bit adder show us the effect of parasitic components. These show why it is so important to simulate with parasitic components. The delay through the 32-bit adder goes from 11.55µs when simulating on the schematic to 15.32µs when simulating the adder on cell level (Verilog code as described in Section 3.7.2). From these, the delay goes up further too 18.68µs when simulating on the entire design.

The energy consumed by the adder is shown in Table F-9. When running at a supply voltage of 300mV, which is what the library is designed for, the adder consumes about 60-70fJ (60×10^{-15}) per calculation (of the ones we have tried). When the same circuit is simulated on the same test bench using a supply voltage of 1.2V, the energy per calculation goes up to 1.6pJ (1.6×10^{-12}), which is more than 20 times the amount consumed by the low power version.

In the current library there are several small errors that need to be fixed manually. This must be done after the synthesizer has put together the cells to a larger design. It was not a time consuming process to do this manually and has therefore not been dealt with. Most of these errors, if not all, can, with some work, be removed if the library cells are designed slightly different.

Chapter 6

6 Conclusion

In this thesis, the design and optimization of a standard cell library for subthreshold have been presented. The results have been found using multiobjective optimization algorithms and custom fitness functions simulating over three temperatures, -20º, 27º and 85º.

The optimization objectives have been timing, static power dissipation and robustness. The best relationship between the objectives has been found, using a custom grid based multiobjective optimization algorithm with Pareto optimality. Because we use high threshold transistors and have a very strict robustness requirement, the cells will become larger and slower, but consume less energy. The threshold voltage of the transistors is 450mV and we use a supply voltage of 300mV. This gives an area of operation well below the threshold voltage.

The cells completed are a NOT, NOR, NAND and an XOR gate. With these cells we can synthesize all kinds of digital designs. The values of the cells were chosen based on extensive simulations and optimization. The final library is robust and consumes very little energy. To use a library with cells for synthesis reduces the production time of a chip with several orders of magnitude, when compared to manual layout.

The library has been tested and verified through synthesis and place and route by the synthesizing of a signed 32-bit adder with carry. The adder has been simulated with and without parasitic components on 1.2V and 300mV in order to compare the results. At a supply voltage of 300mV, the adder is able to run at approximately 50 kHz and consumes about 60fJ per calculation at 27º.

6.1 Further work

This thesis has shown some of the methods for optimizing a library for subthreshold operations. The potential of further work is large.

The optimization algorithms and fitness functions used in the thesis have been customized for the cells optimized. It can be useful to rewrite these to be more generic.

The library shown in the thesis only contains four completed cells. In order to get a good library, several cells could be implemented in the library, such as adder cells, different latches and flip-flops. In order to get a better performing DFF, the topologies described in [38] should be tested. To match them to the rest of the library, it might be advantageous to

simulate them on transistor level in accordance with the performance variables used in Section 3.5.2.

To get an adder in the design, the high-performance 1-bit full adder from [42] and some of the adder topologies given in [43] should be tested with a modified version of the fitness function. It could also be interesting to try other topologies for the XOR, NAND and NOR gates (dynamic logic, pass transistor logic, etc.).

In order to get a complete verified library, several benchmark circuits should be synthesized and simulated with Monte Carlo method in order to compare the performance to other libraries.

Appendix

Appendix A

Design rules and layout guidelines

Standard cells are a group of transistors and interconnect structures that make out a Boolean function or a memory function. Since the logical definition of a cell, only is good for functional simulation, the physical representation must be designed. This is the lowest level of design and is the design that is shipped off to manufacturers. This is the layout level.

If you are not familiar with layout, it can be explained quickly as: The layout is set up in different layers. The lowest level is the p-substrate on which the transistors are “printed”. In this layer there are several areas “doped” with positive or negative particles. Next, there are different metal layers for wiring and layers for VIAs. When drawing the layout, there are several rules that have to be followed and methods for maximizing routing. When a cell is to be used in a library, there are some rules that can be ignored for the cell (e.g. density and minimum area, etc.) and some rules that have to be inserted (e.g. fixed heights, routing grid, etc.).

A.1 Design rules

When designing a layout there are a waste amount of design rules that apply to the different processes. The most important rules to have in mind are the different spacing rules, rules for contacts, wells, line widths and rules for minimum area:

- Minimum distance
  For shapes in the different layers, there are rules for how close shapes in the same layer can be. This is especially something that one has to bear in mind when routing wires between circuits and adding metal shapes to the layout.

- Line widths and area
  The different metals have different rules for their minimum widths and total area. The routing for metal layers has minimum widths.

- Contacts and VIAs
  The contacts and the VIAs do not in itself need to be over the minimum area of the layer it is in, if a VIA goes from M1 to M2, and there is routing in both layers. The area is the total of connected metal in a layer. A contact in a standard cell can be placed anywhere in the cell as long as there are no metal layers over the contact. For a chip, all contacts must be along the chip edges, with no obstructions in layers above.

- N-wells and well contacts
  It is important that wells that ought to be connected have good connection with each other. Wells with different potentials have a minimum distance of more than three times the grid size, wells with the same potential that are not connected directly, have a smaller minimum distance.

A.2 Maximizing Routing

A logic cell for a library must be properly designed in order to obtain optimum place and route. A poorly designed library can result in poor routing [44], bad cell-placement and long running time for the place and route tool. In order to have adequate routing there are several things that must be thought through and considered. The most important one is to set a suitable routing grid. This will be discussed below. Then the second and third steps are to decide the cell-height and cell-widths. There are four main styles of cell placement we have to look at when deciding the cell-height.

- Figure A-1: Single row with variable height. These designs are not efficient for three layers or more, they are usually two-layer design.
- Figure A-2: Single row with fixed height. These designs usually have two or more layers. Cell height is fixed to one value.
- Figure A-3: Single- and double height cells. This design is arranged in double rows and is meant for power sharing.
- Figure A-4: Multiple height cells. This is basically a sea-of-cells that features three or more layers. In this design it is possible to have single, double and multi-height cells as long as the height is a multiple of the minimum height.

Figure A-1: Single row with variable height

Figure A-2: Single row with fixed height

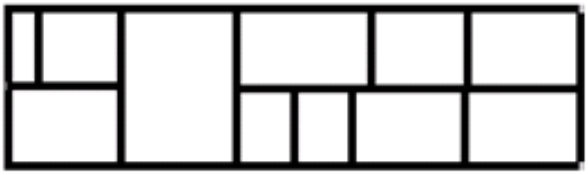


Figure A-3: Single- and double height cells

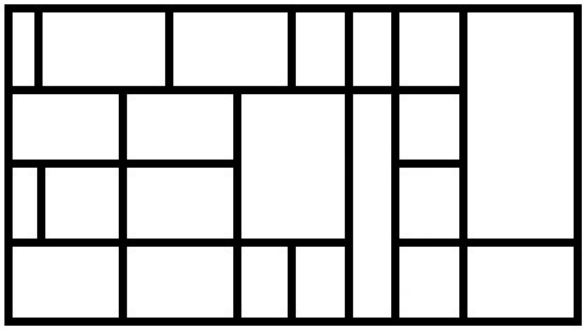


Figure A-4: Multiple height cells

The cell-heights should always be a multiple of the routing grid size. This means that if the spacing between the wires in the grid is X, the height is always a multiple of X, for example 5X or 6X. This also applies for the widths. The difference is that we don't have any special styles for the widths. No matter which style that is chosen for the heights, one can have different widths. This is as long as the widths add up to a multiple of X.

In the library cells for this thesis, "Single- and double height cells" have been used. This gives us a compact design that is easy to rout power and ground to. It is also the easiest design to set up for standard cells in the tools used.

A.3 Routing Grid Pitch

When deciding the size of the routing grid, there are several objectives that have to be taken into account. First, one has to decide what type of minimum size one wants [5, 45]. The three types that are usually used is: line-to-line, via-to-line and via-to-via. A line-to-line grid has the smallest grid size. This grid size is based on the minimum distance between the routing lines and is the grid that has the most compact routing. The main problem with this type of grid is that one cannot use a VIA in the line next to another routing line. This means that every time one uses a VIA, one has to drop a routing line.

The next type, line-to-via, does not have this problem. This is the most compact grid where it is possible to have VIA holes on the line without having to stop using the lines on both sides of the line with the VIA or route around the area the VIA is in.

The last type of grid is the via-to-via. As the name suggests, this is the grid where it is not a problem to put a VIA next to a VIA, without causing proximity violations.

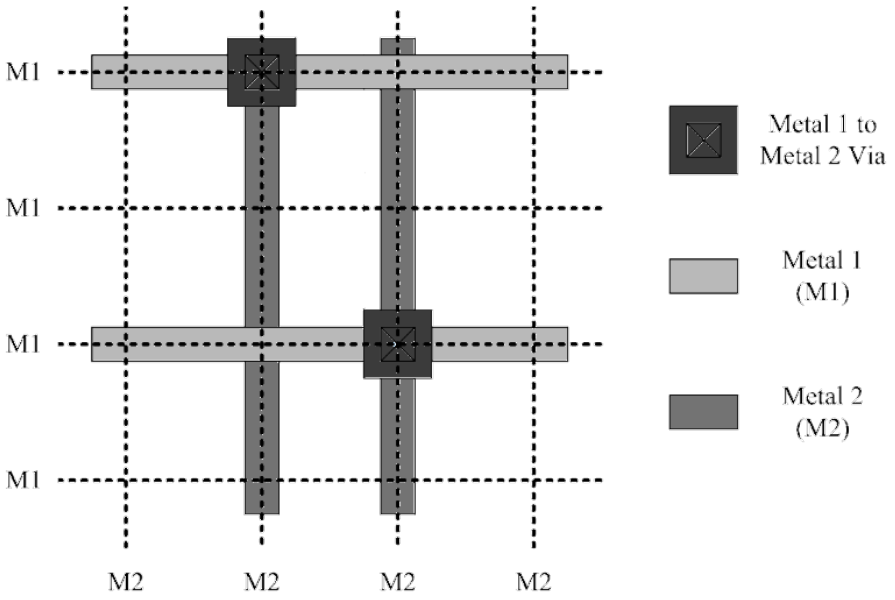


Figure A-5: VIA placement in routing grid

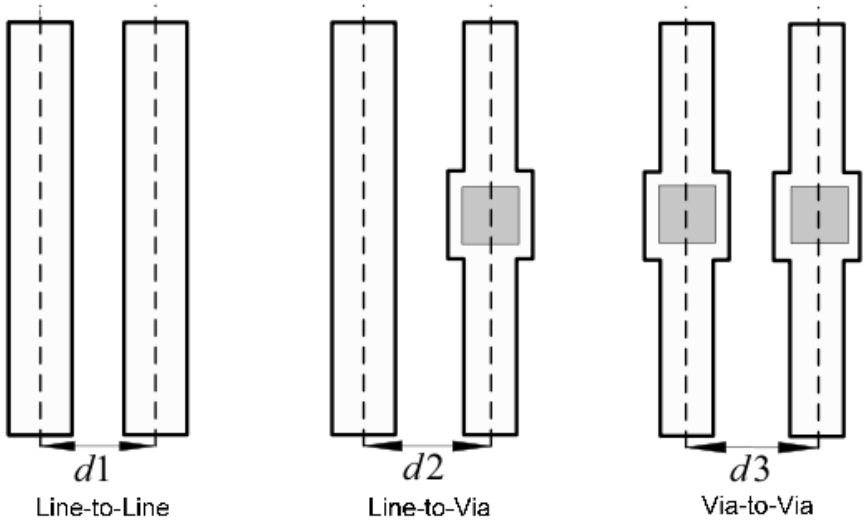


Figure A-6: Wire track centre to centre spacing [5]

The grid size used in this thesis has been set to 0.33 µm. This gives us a line to VIA spacing. The minimum cell width is set to an even multiple of the grid size. This is because some of the wells and "doped" areas get some errors if the distance is less than minimum distance, even if they have the same potential. If a filler cell with a smaller width than the minimum distance has a well area that do not exactly matches the cells on both sides, it will get an error. Because cells need to be aligned, there cannot be cells with an odd multiple of the grid size.

A.4 Fillers and well contacts

Fillers are cells that are set in where there is empty area between standard cells. The fillers have several applications and can set to serve several purposes. The main purpose of a filler cell is to maintain the connections to the power grid where a routing tool does not do the job. Another

important purpose for the filler is that the filler cell makes sure we have contact between the n-wells of same potential. In larger designs there is often a minimum limit of metal density over the total area of the chip. A design cannot utilize 100% of the chip area. This is because a design needs space for improving routing. Filler cells with dummy structures and metal fill are therefore used. Another use for filler cells is to place well contacts around in the design. Any pMOS in an n-well need to have bulk contact to the supply power and any nMOS in a p-well (which is the normal metal in p-substrate process) needs to have bulk contact to ground. The filler cells can therefore be used to place well contacts.

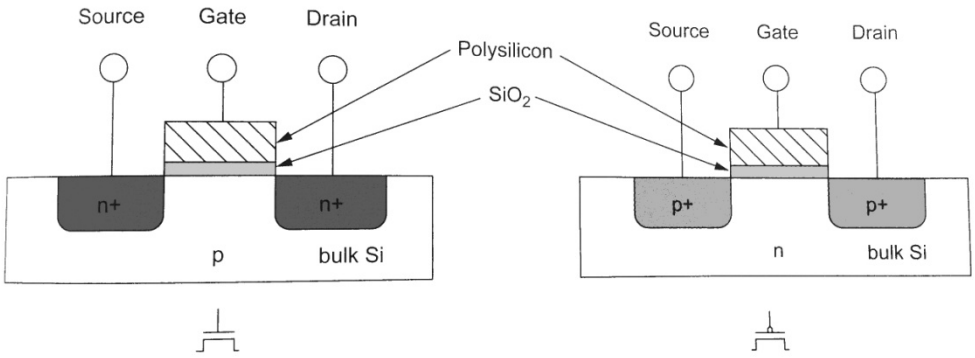


Figure A-7: Cross section of two transistors [4]

Appendix B

Design flow

The workflow diagram shows the order and correspondence between the different tools used in the library. It also shows important order of some of the sub tools (marked with light gray) and the files that are generated (dark gray). The black boxes are names on specific tools that automatically generate files or layouts and the rest is files, schematics and layout that are designed. The design flow is described in [40].

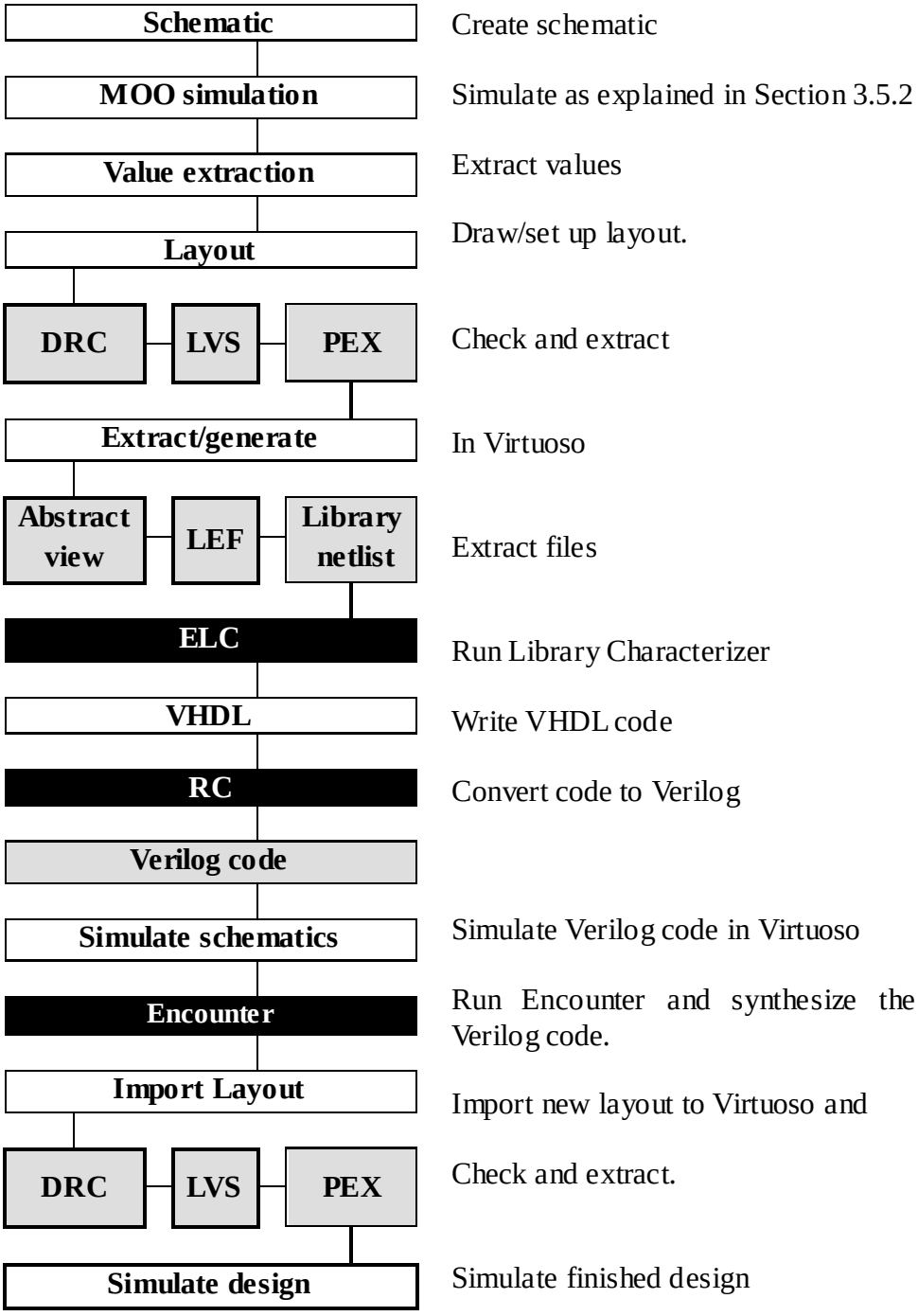


Figure B-8: Design flow diagram

Appendix C

Schematics

C.1 NOT gate

I	Out
0	1
1	0

Table C-1: Truth table of a NOT gate

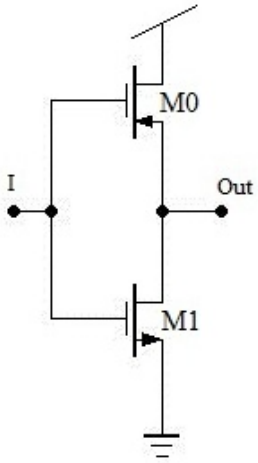


Figure C-9: Schematic of a NOT gate

An NOT gate consists of one nMOS and one pMOS transistor. When the signal on the input is high, the pMOS will stop conducting voltage from the supply voltage, and the nMOS will start conducting between the output and ground. When the signal in is low, the pMOS will conduct the supply voltage to the output and the nMOS will stop conducting.

C.2 NAND gate

I1	I2	Out
0	0	1
0	1	1
1	0	1
1	1	0

Table C-2: Truth table of a NAND gate

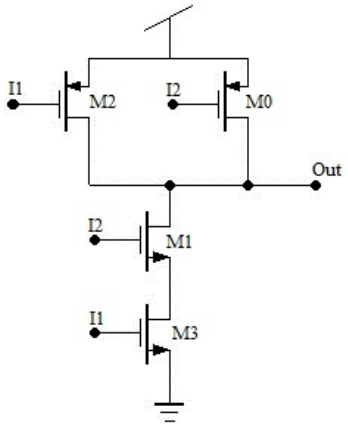


Figure C-10: Schematic of a NAND gate

A NAND gate consists of two pMOS transistors in parallel connected to

the supply voltage and two nMOS transistors in series connected to ground. The output will be high if at least one of the pMOS transistors receive a low signal and it will only go low if both the nMOS transistors are high.

C.3 NOR gate

I1	I2	Out
0	0	1
0	1	0
1	0	0
1	1	0

Table C-3: Truth table of a NOR gate

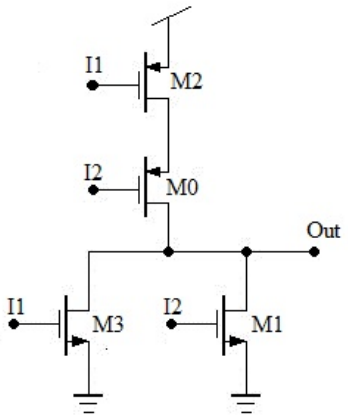


Figure C-11: Schematic of a NOR gate

A NOR gate consists of two pMOS transistors in series connected to the supply voltage and two nMOS transistors in parallel connected to ground. The output will be low if at least one of the nMOS transistors receive a high signal and it will only go high if both the pMOS transistors are low.

C.4 XOR gate

I1	I2	Out
0	0	0
0	1	1
1	0	1
1	1	0

Table C-4: Truth table of a XOR gate

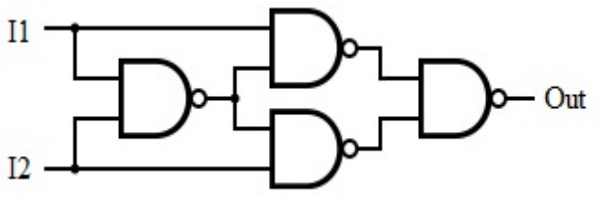


Figure C-12: Schematic of an XOR gate

The XOR gate is an exclusive OR gate. It consists of four NAND gates. The output of this gate will go high only if one of the inputs is high ($I_1 \neq I_2$).

C.5 D-latch

E	D	Q	$\overline{Q}$
0	X	Q_{prew}	$\overline{Q_{prew}}$
1	0	0	1
1	1	1	0

Table C-5: Truth table of a D-LATCH

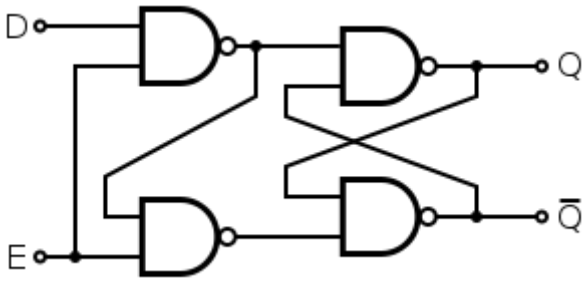


Figure C-13: Schematic of a D-LATCH

The D-latch can be viewed as a simple memory circuit. The circuit is transparent when the signal D is high, and it memorizes the last output when D is low ('X' denotes a "Do not care" condition.) [46].

C.6 Delay flip-flop

Clock	D	Q_{next}	Q
Falling edge	0	0	X
Falling edge	1	1	X

Table C-6: Truth table of a DFF

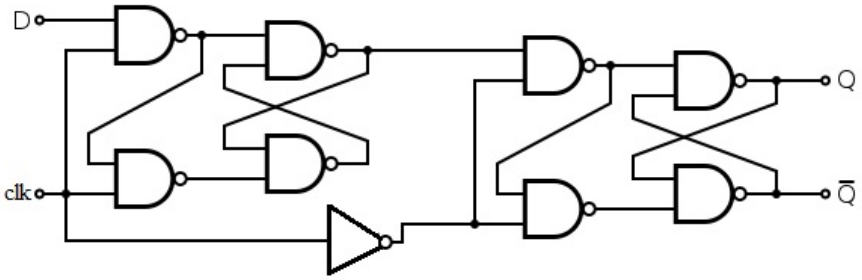


Figure C-14: Schematic of a DFF

A delay flip-flop takes the value of D and memorizes when the clock goes high, the output will then present it when the clock goes low [46].

Appendix D

Simulation files

D.1 NOT gate SCS-file

```
// Generated for: spectre
// Generated on: Dec 13 19:32:22 2011
// Design library name: my_testlib
// Design cell name: inv_hvt
// Design view name: schematic
simulator lang=spectre
global 0 vdd!
parameters MynL=100n MynW=120n MypL=100n MypW=120n vdd=300m
Qsig=0.8*vdd Asig=0.2*vdd
include "inv_params.scs"
include "mc_test3.scs"
alter0 altergroup{
    parameters Asig=0.2*vdd Qsig=0.8*vdd
}
alter1 altergroup{
    parameters Asig=0.8*vdd Qsig=0.2*vdd
}
// Library name: my_testlib
// Cell name: inv_hvt
// View name: schematic
M0 (Q A vdd! vdd!) pch_hvt_mac l=MypL w=MypW*1 multi=(1) nf=1 sd=260.0n
\
    ad=((1-int(1/2)*2)*(2.3e-07+((1-1)*2.6e-07)/2+0)+(1+1-
    int((1+1)/2)*2)*((1/2)*2.6e-07))*MypW \
    as=((1-int(1/2)*2)*(2.3e-07+((1-1)*2.6e-07)/2+0)+(1+1-
    int((1+1)/2)*2)*(2.3e-07+2.3e-07+(1/2-1)*2.6e-07+0+0))*MypW \
    pd=(1-int(1/2)*2)*((2.3e-07+((1-1)*2.6e-
    07)/2+0)*2+(1+1)*MypW)+(1+1-int((1+1)/2)*2)*(((1/2)*2.6e-
    07)*2+1*MypW) \
    ps=(1-int(1/2)*2)*((2.3e-07+((1-1)*2.6e-
    07)/2+0)*2+(1+1)*MypW)+(1+1-int((1+1)/2)*2)*((2.3e-07+2.3e-07+(1/2-
    1)*2.6e-07+0+0)*2+(1+2)*MypW)\
    nrd=(1-int(1/2)*2)*(1.3e-07*1.3e-07/(1.3e-07+1.3e-07*(1-
    1))/MypW)+(1+1-int((1+1)/2)*2)*(1.3e-07/1/MypW) \
    nrs=(1-int(1/2)*2)*(1.3e-07*1.3e-07/(1.3e-07+1.3e-07*(1-
    1))/MypW)+(1+1-int((1+1)/2)*2)*(1.3e-07*1.3e-07*1.3e-07/(1.3e-
    07*1.3e-07*(1-2)+1.3e-07*(1.3e-07+1.3e-07))/MypW) \
    sa=1/(0.0+1.0/(2.3e-07+(0.5*MypL)+0*(2.6e-07+MypL)))-(0.5*MypL) \
    sb=1/(0.0+1.0/(2.3e-07+(0.5*MypL)+0*(2.6e-07+MypL)))-(0.5*MypL) \
    sca=0 scb=0 scc=0 mismatchflag=1
    M1 (Q A 0 0) nch_hvt_mac l=MynL w=MynW*1 multi=(1) nf=1 sd=260.0n \
    ad=((1-int(1/2)*2)*(2.3e-07+((1-1)*2.6e-07)/2+0)+(1+1-
    int((1+1)/2)*2)*((1/2)*2.6e-07))*MynW \
    as=((1-int(1/2)*2)*(2.3e-07+((1-1)*2.6e-07)/2+0)+(1+1-
    int((1+1)/2)*2)*(2.3e-07+2.3e-07+(1/2-1)*2.6e-07+0+0))*MynW \
    pd=(1-int(1/2)*2)*((2.3e-07+((1-1)*2.6e-
    07)/2+0)*2+(1+1)*MynW)+(1+1-int((1+1)/2)*2)*(((1/2)*2.6e-
    07)*2+1*MynW) \
    ps=(1-int(1/2)*2)*((2.3e-07+((1-1)*2.6e-
    07)/2+0)*2+(1+1)*MynW)+(1+1-int((1+1)/2)*2)*((2.3e-07+2.3e-07+(1/2-
    1)*2.6e-07+0+0)*2+(1+2)*MynW)\
```

```
    nrd=(1-int(1/2)*2)*(1.3e-07*1.3e-07/(1.3e-07+1.3e-07*(1-
    1))/MynW)+(1+1-int((1+1)/2)*2)*(1.3e-07/1/MynW) \
    nrs=(1-int(1/2)*2)*(1.3e-07*1.3e-07/(1.3e-07+1.3e-07*(1-1))/MynW)
    +(1+1-int((1+1)/2)*2)*(1.3e-07*1.3e-07*1.3e-07/(1.3e-07*1.3e-07*(1-
    2)+1.3e-07*(1.3e-07+1.3e-07))/MynW) \
    sa=1/(0.0+1.0/(2.3e-07+(0.5*MynL)+0*(2.6e-07+MynL)))-(0.5*MynL) \
    sb=1/(0.0+1.0/(2.3e-07+(0.5*MynL)+0*(2.6e-07+MynL)))-(0.5*MynL) \
    sca=0 scb=0 scc=0 mismatchflag=1
V2 (Q 0) vsource dc=Qsig type=dc
V1 (vdd! 0) vsource dc=vdd type=dc
V0 (A 0) vsource dc=Asig type=dc
simulatorOptions options reltol=1e-5 gmin=1e-18 abstol=1e-6 iabstol=1e-15
dcOp dc force=node homotopy=gmin maxiters=150  maxsteps=10000
dc dc hysteresis=yes param=temp values=[-20 27.5 85] force=node
homotopy=gmin maxiters=150 maxsteps=10000
saveOptions options save=nooutput
```

D.2 NOT gate MDL-file

```
alias measurement dc2
  {
   run dc
   real ioff1   = -I(M0:1)@-20
   real ioff2   = -I(M0:1)@27.5
   real ioff3   = -I(M0:1)@85
   real ion1    = I(M1:1)@-20
   real ion2    = I(M1:1)@27.5
   real ion3    = I(M1:1)@85
   print fmt("%e %e %e %e %e %e\n", ion1, ion2, ion3, ioff1, ioff2,
                                                    ioff3) addto="mc1.txt"
  }
alias measurement dc1
  {
   run dc
   real ioff1   = I(M1:1)@-20
   real ioff2   = I(M1:1)@27.5
   real ioff3   = I(M1:1)@85
   real ion1    = -I(M0:1)@-20
   real ion2    = -I(M0:1)@27.5
   real ion3    = -I(M0:1)@85
   print fmt("%e %e %e %e %e %e\n", ion1, ion2, ion3, ioff1, ioff2,
                                                    ioff3) addto="mc1.txt"
  }
int nummcruns = 30
foreach data_WL1_1 {
    run montecarlo(numruns=30, seed=1, variations = 'all, firstrun = 1,
                                                     donominal = 'yes )
    {
      run dc1
    }
    }
run alter1
foreach data_WL1_1 {
    run montecarlo(numruns=30, seed=1, variations = 'all, firstrun = 1,
                                                     donominal = 'yes )
    {
      run dc2
    }
     }
```

D.3 Grid based multiobjective optimization

Written by: Hans K.O. Berge, hansbe@ifi.uio.no

```
function res=step_opt4(func);
% Multi-objective optimization
% Start with grid, step from pareto points.
% (c) Hans K.O. Berge, hansbe@ifi.uio.no
%func=@sim_inv;
lb=[0.12 0.12 0.1 0.1];
ub=[10 5 4 4];
pdims=length(lb);
%options.initsteps= 5e-3*1*[1 1 1 1];
options.minstep = 5e-3*[1 1 1 1];
options.initsteps=ongridc((ub-lb)/2,options.minstep);
MAXEVALS=20000; % avoids too many evaluations which could result in full
memory
MAXPASSES=3; % max iterations on each step size

division = (ub-lb)/10; % N=10, NxNxNxN initial grid

if exist('opt.mat','file')
 % assume we are continuing
 disp 'loading opt.mat to continue simulations'
 load('opt.mat')
 steps=options.initsteps
else %initalize parameters
% make vectors:
cvec=cell(1,pdims);
for idim=1:pdims

cvec{idim}=ongrids([lb(idim):division(idim):ub(idim)],options.minstep(idi
m));
   cvec{idim}=max(cvec{idim},lb(idim));
   cvec{idim}=min(cvec{idim},ub(idim));
   cvec{idim}=unique(cvec{idim});
end
params=gridpoints(cvec);
disp(sprintf('Running %d initial simulations',size(params,1)));
funevals = func(params); % <-- Simulations

front = paretofront(funevals);
ffunevals=funevals(front,:);
funevals=funevals(~front,:);
fparams=params(front,:);
params=params(~front,:);

save('opt.mat','fparams','ffunevals', 'params','funevals')
end

% intial steps
steps=options.minstep.*(2.^floor(log2(division./options.minstep)))

allsteps=acombine([-1:1]'*ones(1,pdims));       % steps from current
front
allsteps=setdiff(allsteps,[0 0 0 0],'rows');           % unneccessary
step

%step on grid
%steps=ceil(steps./options.minstep).*options.minstep;
passes=0;
min_step_pass=0;
while min_step_pass<=MAXPASSES

% make new points
step_matrix=prodrow(allsteps,steps);
sz_sm = size(step_matrix,1)
sz_fp = size(fparams,1);
```

```
np=zeros(sz_sm*sz_fp, pdims);
%size(np)
%size(plusrow(step_matrix,fparams(1,:)),1)
tic;
for ii=1:size(fparams,1)
  np(ii*sz_sm+[1:sz_sm],:)=plusrow(step_matrix,fparams(ii,:));
end
size(np)
t0=toc
for idim=1:pdims
np(:,idim)=min(np(:,idim),ub(idim)); % upper bound
np(:,idim)=max(np(:,idim),lb(idim)); % lower bound
%put on grid
% should now be on grid.
%np(:,idim)=round(np(:,idim)/options.minstep(idim))*options.minstep(idim)
;
end
%avoid duplicate evaluations

np=unique(np,'rows');
np=setdiff(np,params,'rows');
np=setdiff(np,fparams,'rows');

%evalution happens in the below clause
if ~isempty(np)
 % MAXEVALS reason -- avoid too large memory consumption
 eval_sz=min(size(np,1),MAXEVALS);
 eval_done=0;
 iter=0;
 while (eval_sz>0)
 disp(sprintf('Running %d out of %d remaining
simulations',eval_sz,size(np,1)-eval_done));
 newfunevals=func(np([1:eval_sz]+iter*MAXEVALS,:));  % <-- Simulation
 ffunevals=[ffunevals; newfunevals];
 iter=iter+1; eval_done=eval_done+eval_sz;
 eval_sz=min(size(np,1)-iter*MAXEVALS,MAXEVALS);
 end
 %steps=options.initsteps;
 passes=passes+1;
else
 passes=MAXPASSES;
end

% Faster out of loop  ?
%ffunevals=[ffunevals; newfunevals];
tic
fparams=[fparams; np];
front = paretofront(ffunevals);
funevals=[funevals; ffunevals(~front,:)];
params  =[params; fparams(~front,:)];
ffunevals=ffunevals(front,:);
fparams=fparams(front,:);
save('opt.mat','fparams','ffunevals', 'params','funevals')
toc

if passes>=MAXPASSES;
%reduce step
steps=ongrid(steps/2, options.minstep)
passes=0;
end


if all(steps<=options.minstep)
 min_step_pass=min_step_pass+1;
end
```

```
%steps=ceil(steps./options.minstep).*options.minstep;
%front=paretofront(funevals);
%try
 figure(1);
 clf;
 plot3(ffunevals(:,1),ffunevals(:,2),ffunevals(:,3),'.')
% pause(1);
 %display
 numinfront=size(ffunevals,1)
 drawnow;
%catch me
%end

end % while

res=[fparams ffunevals];


% helper functions
function r=vcombine(v1,v2)
% All combinations of two row vectors (of equal length)
if length(v1)>1
 rt = vcombine(v1(2:end),v2(2:end));
 r = [ones(size(rt,1),1)*v1(1) rt; ones(size(rt,1),1)*v2(1) rt];
else
 r=[v1;v2];
end %if
%end %combine()

function a=acombine(arr)
a=[];
for i=1:size(arr,1)-1
 for j=i+1:size(arr,1)
   a=[a; vcombine(arr(i,:),arr(j,:))];
 end
end


% operate on each column of a with scalar of row-vector
function r=prodrow(a,row)
for i=1:size(row,2)
r(:,i)=a(:,i)*row(i);
end
%end %prodrow

function r=plusrow(a,row)
for i=1:size(row,2)
r(:,i)=a(:,i)+row(i);
end
%end %plusrow

function r=ongrid(a,minstep)
pdims=size(a,2);
for idim=1:pdims
 r(:,idim)=round(a(:,idim)/minstep(idim))*minstep(idim);
end

function r=ongrids(a,minstep)
% for scalar
 r=round(a/minstep)*minstep;


function r=ongridc(a,minstep)
pdims=size(a,2);
for idim=1:pdims
 r(:,idim)=ceil(a(:,idim)/minstep(idim))*minstep(idim);
end
```

```
function a=gridpoints(cvec)
% function a=gridpoints(cvec)
%
% Returns: Array of points in a grid defined by vectors in cvec.
%
% Inputs:
%  cvec is a cell row vector of row vectors
%
% (c) Hans K.O. Berge, hansbe@ifi.uio.no
if (size(cvec,1)>1 || ~iscell(cvec))
   error(['cvec is a cell row vector of row vectors with grid division on
axis.\n' ...
          'E.g.: cvec={[-1 0 1] [-1 0 1] ...}']);
end
pdims=size(cvec,2);
g=cell(1,pdims);
[g{:}] = ndgrid(cvec{:});
siz=1;
for i=1:pdims
   siz=siz*size(cvec{i},2);
end
a=zeros(siz,pdims);
for idim=1:size(cvec,2)
   a(:,idim)=reshape(g{idim},siz,1);
end
```

D.4 Fitness function for a NOT gate

```
function [resvec] = sim_inv(parameters)
tic
% Sett lengths and widths
WP = parameters(:,1);
LP = parameters(:,3);
WN = parameters(:,2);
LN = parameters(:,4);

% Calculate Cap
for  i=1:size(parameters,1)
    H=max(WP(i)+WN(i)+0.3,1.28);
    B=max(max(LP(i),LN(i))+0.1,0.64);
    res.cap(i)=(H+B)*15*0.2*1e-15;
end

wd = [pwd '/'];
maxproc=25; % Max number of processes
fidxname = '/hom/mshaugla/chip/inv_test_dmv.mat'; % Name of variable
file
pars = squeeze(parameters);
popsize = size(pars,1);

% Set maxsize of parameterfiles
maxproc=min(maxproc,max(1,round(popsize/5)));
if (maxproc>popsize)
    maxproc=popsize;
end

% Split parameters in chunks
popchunks = [0:ceil(popsize/maxproc):popsize];
if popchunks(end)~=popsize
    popchunks =[popchunks popsize];
end
maxproc=length(popchunks)-1;

sims=30; % Number of Monte Carlo runs
temps=3; % Number of temperatures simulated
```

```
% Write parameter file
fid_param = fopen([wd 'inv_params.scs'],'w');
for i=1:maxproc
    fprintf(fid_param, '\ndata_WL1_%d paramset {\nMypW MynW MypL
MynL\n',i);
    fprintf(fid_param, '%.4fu %.4fu %.4fu %.4fu\n',
pars((popchunks(i)+1):popchunks(i+1),:)');
    fprintf(fid_param, '}');
end
fclose(fid_param);

% delete old output file:
cmd=['rm ' wd 'mc*.txt'];
system(cmd);

%compute values
simcmd = 'nice -15 spectremdl -warn +lqt 0 +error -log -info -note -
batch %s -design %s & echo $!';

for i=1:maxproc
    started=0;

while ~started
        % start spectre ..
        mdlfile = [wd sprintf('inv_%d.mdl', i)]; % Spectre simulation
settup
        scsfile = [wd 'inv.scs']; % Spectre netlist
        cmd = sprintf(simcmd, mdlfile, scsfile);
        [ss,o]=system(cmd);
        try
             pid(i)=str2num(o);
             started = 1;
        catch
            disp(sprintf('Unexpected fail while starting process %d',
                                                                  i));
            disp(o)
            disp('Retrying')
            pause(0.5)
        end
    end
end
while ~isempty(pid)
    [ss,o]=system(sprintf('ps -p %d |grep %d', pid(1), pid(1)));
    while length(o>2)
        pause(0.2)
        [ss,o]=system(sprintf('ps -p %d |grep %d', pid(1), pid(1)));
    end

    pid = pid(2:length(pid));
end
%Calculate values
for i=1:maxproc
    datafile = [wd sprintf('mc%d.txt',i)]; % Read datafiles
    failed=1;
    while failed
      fid = fopen(datafile, 'r');
      if fid~=-1
          failed = 0;
      else
        disp(['Failed in opening ' datafile])
        pause(0.1)
      end
    end
    cs = popchunks(i+1)-popchunks(i); % Number of runs
    data= fscanf(fid, '%e',[temps*2,(sims+1)*cs*2])';
    fclose(fid);
    ffi=[1:(sims+1)]; % Number of values on first input value
    ff2=(sims+1)*cs;  % Number of values on second input value
```

```
    for ff=1:cs
        ffs=(ff-1)*(sims+1);
        res.out0=data(ffs+ffi,:);  % Read out values on first
input values
        res.out1=data(ffs+ffi+ff2,:);     % Read out values on
second input values
        % Rearrange values after type
        Ioff0=res.out0(1:(sims+1),[4 5 6]);
        Ioff1=res.out1(1:(sims+1),[4 5 6]);
        ioff=[Ioff0 Ioff1];
        Ion0=res.out0(1:(sims+1),[1 2 3]);
        Ion1=res.out1(1:(sims+1),[1 2 3]);
        ion=[Ion0 Ion1];
         % Calculate time
        res.t(ff+popchunks(i))=sqrt(mean(mean((res.cap(
                              ff+popchunks(i))./(ion-ioff)).^2)));
         % Calculate ioff
        res.leak(ff+popchunks(i))=mean(mean(ioff));
         % Calculate robustness
        x0t1=std(log10(Ioff0(:,1)/Ion0(:,1)));
        x0t2=std(log10(Ioff0(:,2)/Ion0(:,2)));
        x0t3=std(log10(Ioff0(:,3)/Ion0(:,3)));
        x1t1=std(log10(Ioff1(:,1)/Ion1(:,1)));
        x1t2=std(log10(Ioff1(:,2)/Ion1(:,2)));
        x1t3=std(log10(Ioff1(:,3)/Ion1(:,3)));
        y0t1=mean(log10(Ioff0(:,1)/Ion0(:,1)));

        y0t2=mean(log10(Ioff0(:,2)/Ion0(:,2)));
        y0t3=mean(log10(Ioff0(:,3)/Ion0(:,3)));
        y1t1=mean(log10(Ioff1(:,1)/Ion1(:,1)));
        y1t2=mean(log10(Ioff1(:,2)/Ion1(:,2)));
        y1t3=mean(log10(Ioff1(:,3)/Ion1(:,3)));
        res.re_wc(ff+popchunks(i))=max(1./[x0t1./y0t1 x0t2./y0t2
                    x0t3./y0t3 x1t1./y1t1 x1t2./y1t2 x1t3./y1t3]);
    end
end
res.simt1 = toc;
res      % Prit values to screen.
%Return value matrix from function.
resvec =  [res.t' res.re_wc' res.leak'];
```

Appendix E

Library cell layout

The library cells' layout is presented below. All the cells have been built to fit easy together and to be simple to route in and out from. The actual library cells can be obtained from the nano-group at Ifi, at UiO.

E.1 NOT gate

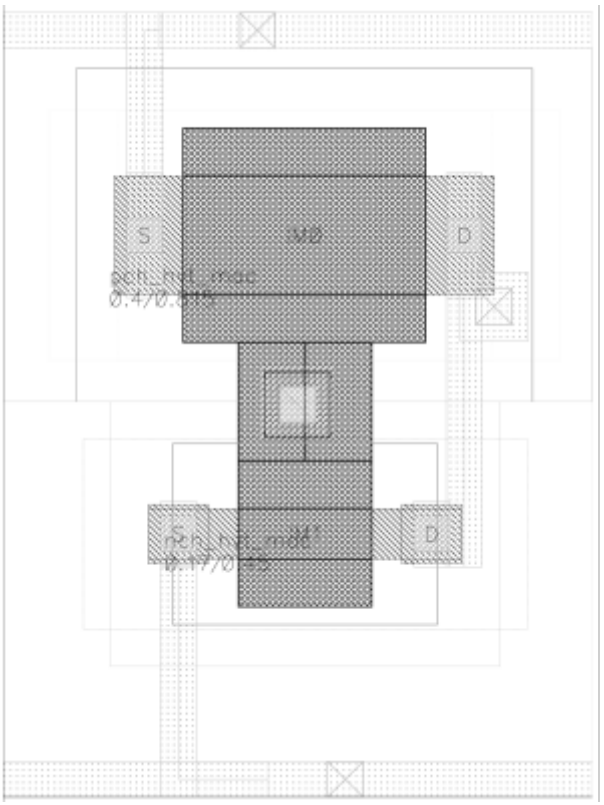


Figure E-15: Layout of a NOT-gate

E.2 NAND gate

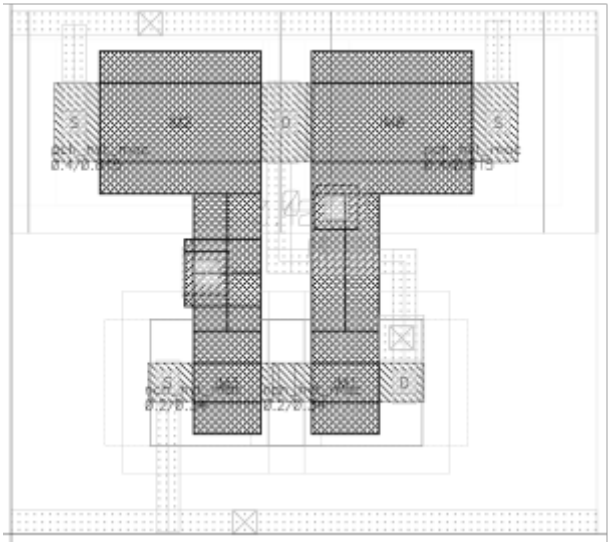


Figure E-16: Layout of a NAND-gate

E.3 NOR gate

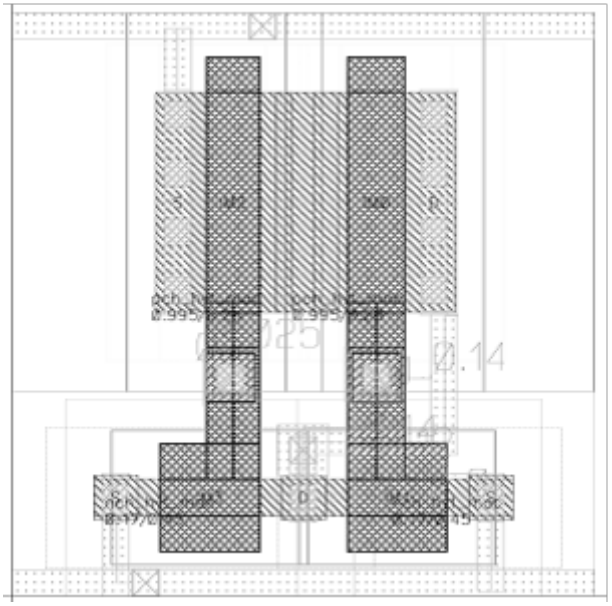


Figure E-17: Layout of a NOR-gate

E.4 XOR gate

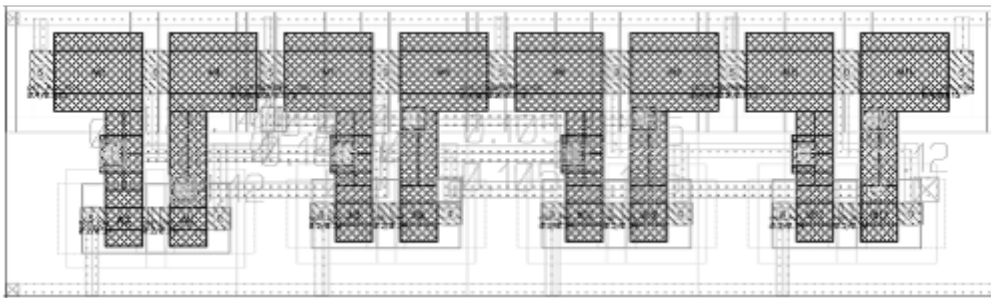


Figure E-18: Layout of an XOR-gate

E.5 D-LATCH

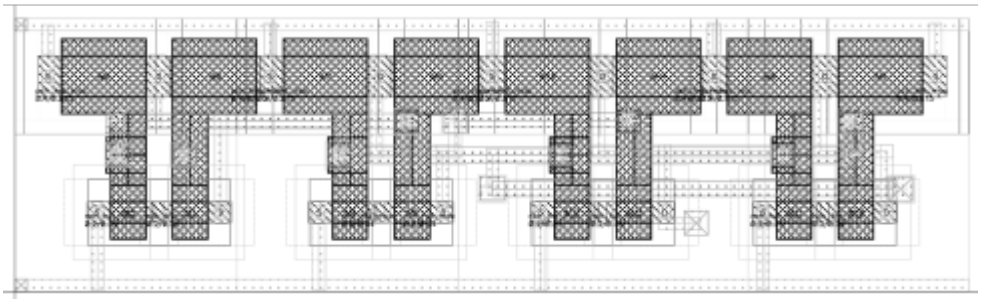


Figure E-19: Layout of a D-LATCH

E.6 DFF

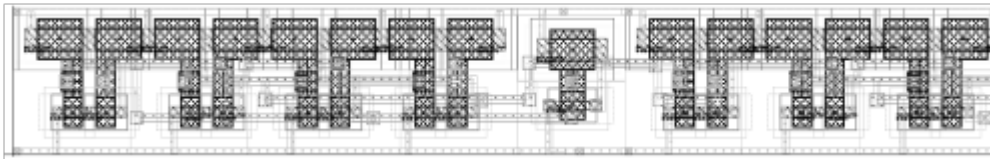


Figure E-20: Layout of a Delay Flip-Flop

E.7 Filler cells

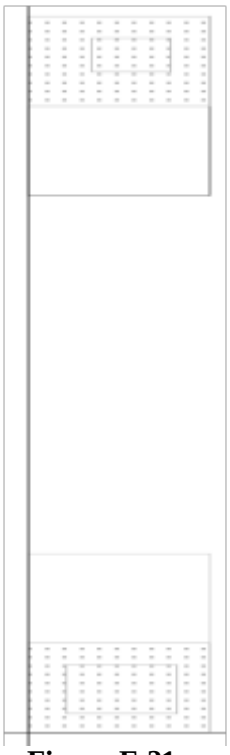


Figure E-21: Layout of an empty filler cell

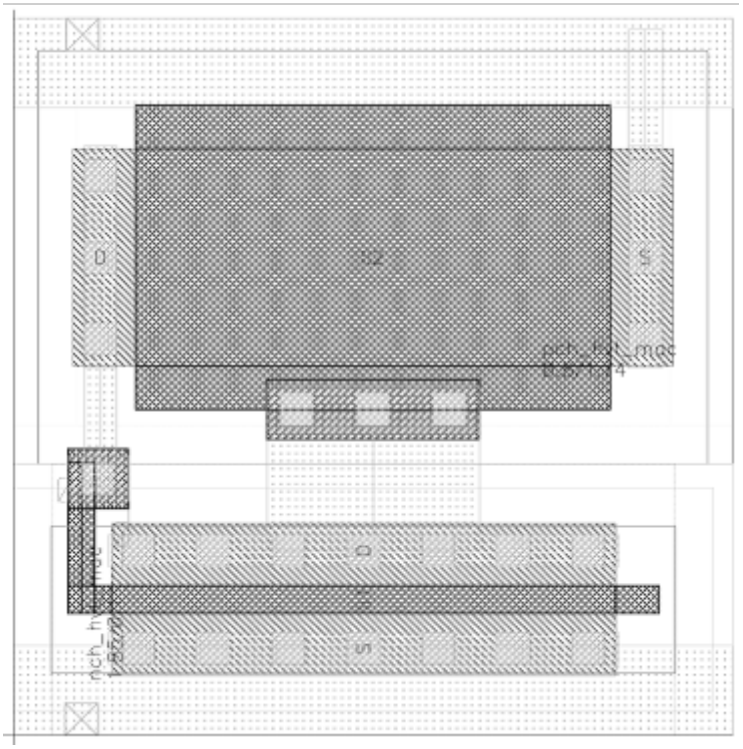


Figure E-22: Layout of a filler with transistors used as decoupling capacitor

Figure E-23: Layout of a filler cell with well contacts

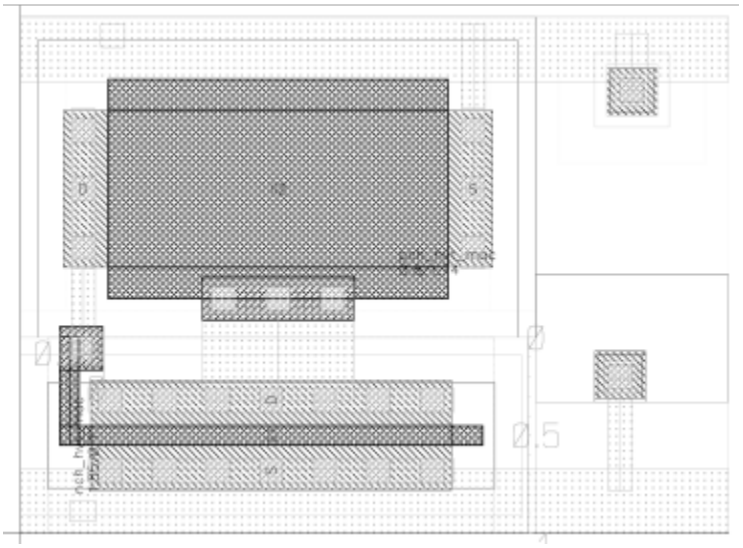
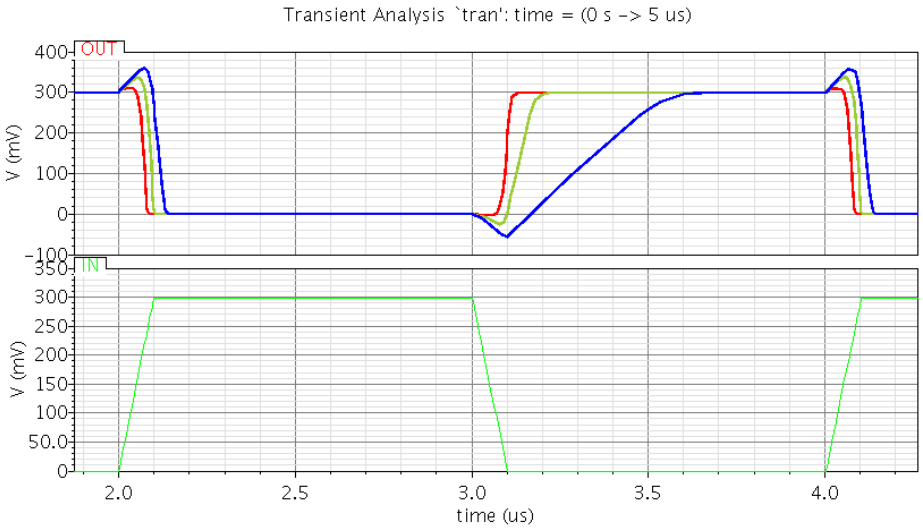


Figure E-24: Layout of a filler with transistors used as decoupling capacitor and well contact

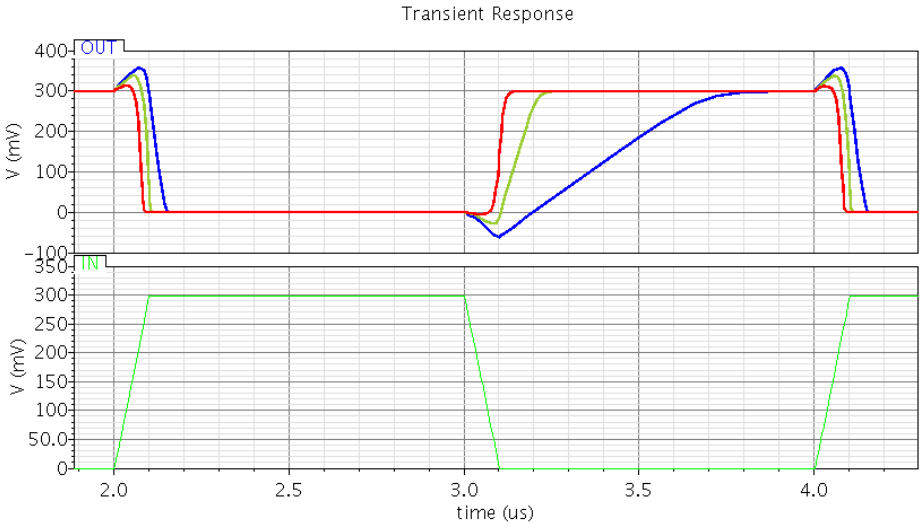
Appendix F

Final simulation results

F.1 Results from the NOT gate



Graph F-1: Delay in a NOT gate without parasitic



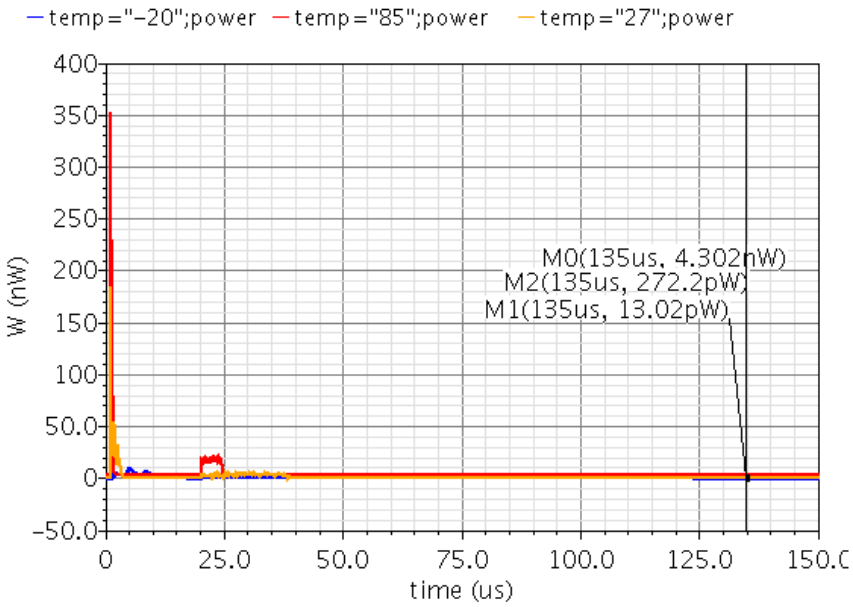
Graph F-2: Delay in a NOT gate with parasitic

	Inv1 (-20◦)	Inv1 (27◦)	Inv1 (85◦)	Inv2 (-20◦)	Inv2 (27◦)	Inv2 (85◦)
Delay L→H	302 ns	89 ns	48 ns	391 ns	106 ns	52 ns
Delay H→L	63 ns	42 ns	23 ns	72 ns	46 ns	26 ns
Energy per switch	38 aJ	37 aJ	40 aJ	52 aJ	49 aJ	51 aJ
Static power (H)	129 fW	945 fW	13.86 pW	129 fW	945 fW	12.86 pW
Static power (L)	94 fW	181 fW	1.96 pW	94 fW	181 fW	1.96 pW

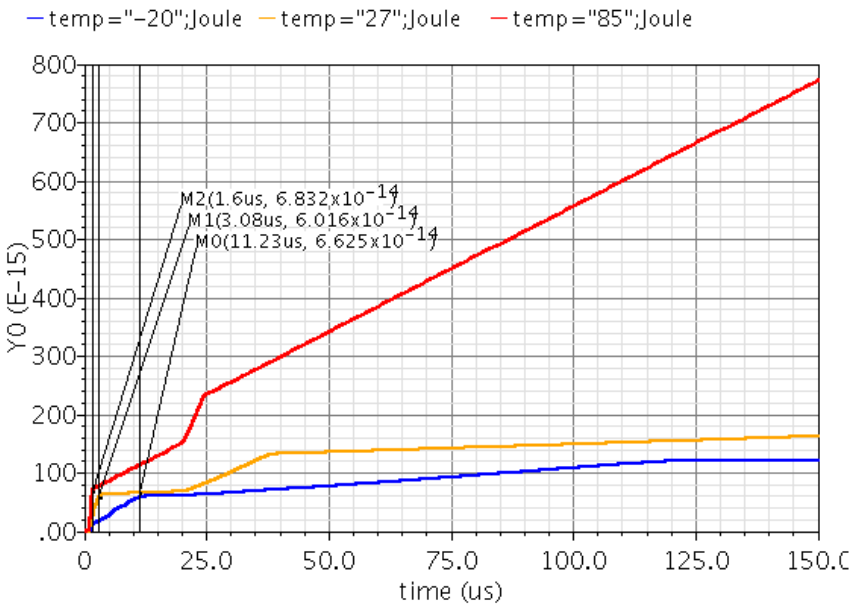
Table F-7: Value sets from the simulation of a NOT gate

- **Inv1:** Simulation without parasitic components. Se Graph F-1.
- **Inv2:** Simulation with parasitic components. Se Graph F-2.
- **Delay L→H:** Delay between the time when the input reaches 50% vdd to the output reaches 50% vdd, when the input goes high and the output goes from vdd→ground.
- **Delay H→L:** Delay between the time when the input reaches 50% vdd to the output reaches 50% vdd, when the input goes low and the output goes from ground→vdd.
- **Static power (H):** Static power when the output is in its high state.
- **Static power (L):** Static power when the output is in its low state.

F.2 32-bit Adder

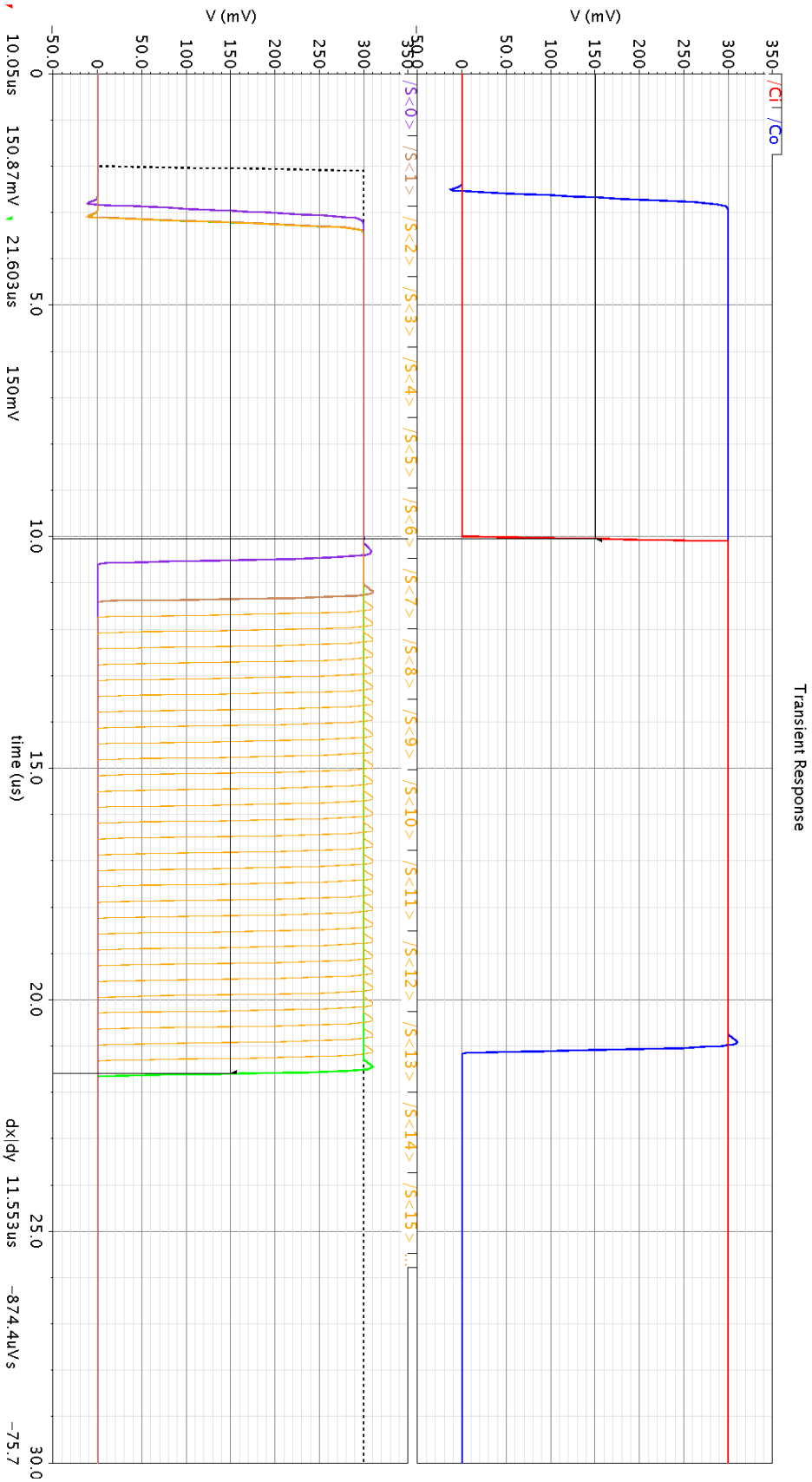


Graph F-3: Power in 32-bit adder



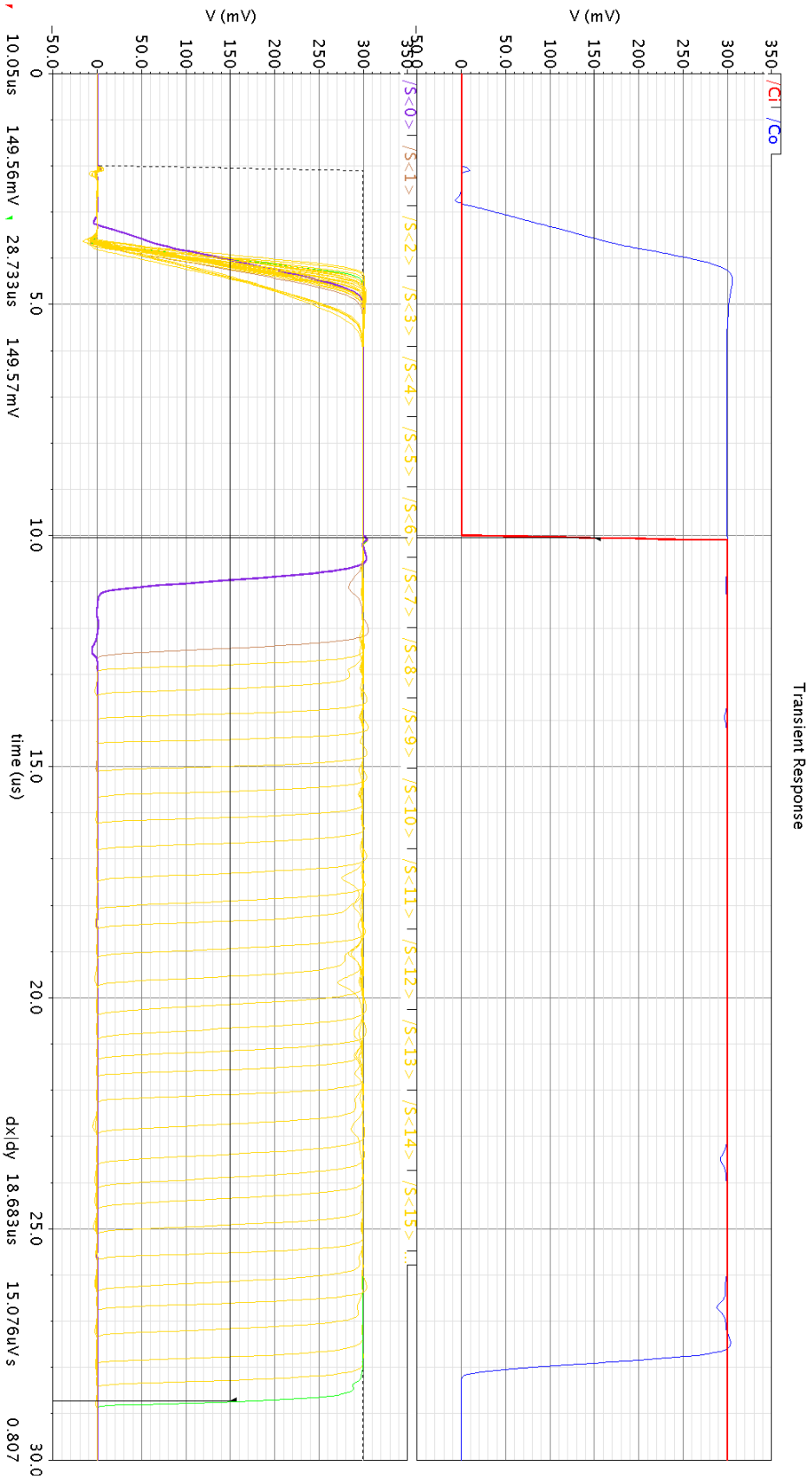
Graph F-4: Energy used in 32-bit adder

The results from Graph F-3 and Graph F-4 have been extracted from the same type of simulation as. Just with three temperatures and over a larger time span.



Graph F-5: Signals from 32-bit adder simulated without extracted parasitic

Graph F-6: Signals from 32-bit adder simulated with extracted parasitic on the cell-level, and without extracted parasitic on the interconnect between cells



Graph F-7: Signals from 32-bit adder simulated with extracted parasitic on the design-level

	Cap	Res
Cell level	27 896	17 957
Design level	82 100	28 050

Table F-8: Number of parasitic components in the 32-bit adder

Temp	Ripple 1	Energy 1	Ripple2	Energy 2
-20.00	103.5 µs	66 fJ	25.66 ns	1.5 pJ
27.00	18.68 µs	60 fJ	27.97ns	1.6 pJ
85.00	4.646 µs	68fJ	30.97ns	1.7 pJ

Table F-9: Comparison between high and low supply voltage on the 32-bit adder

Appendix G

Synthesized design

G.1 Synthesized schematic of a 32-bit Adder

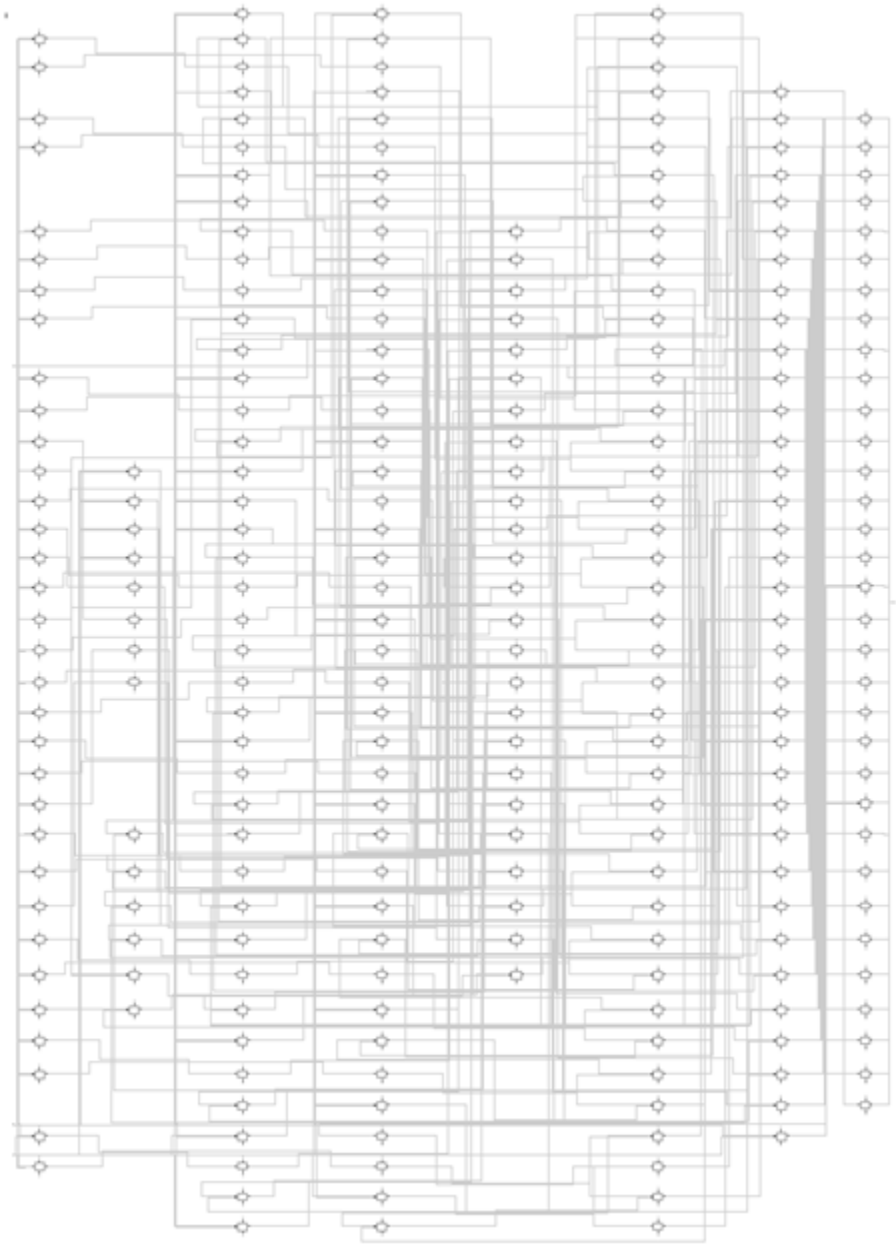


Figure G-25: Schematic of a synthesized 32-bit adder

Acronyms

ASIC	Application-Specific Integrated Circuit
BTBT	Band-To-Band Tunneling
CMOS	Complementary Metal–Oxide Semiconductor
DEF	Design Exchange Format
DRC	Design Rule Checker
DUT	Device Under Test
ELC	Encounter Library Characterization
encounter	Cadence Encounter RTL Compiler
gamultiobj	Genetic Algorithm Multi Objective
HDL	Hardware Description Language
HS	High Speed
IC	Integrated Circuit
LEF	Library Exchange Format
LP	Low Power
LVS	Layout Versus Schematic
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
MOO	Multiobjective Optimization
NM	Noise Margin
NMH	Noise Margin High
NML	Noise Margin Low
nMOS	N-channel MOSFET
NOM	Nominal
PEX	Parasitic Extraction
pMOS	P-channel MOSFET
RF	Radio Frequency
RMS	Root Mean Square
TTM	Time-To-Marked
ULV	Ultra Low Voltage
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration
X-D	X-Dimensional

Bibliography

[1] Hanson, S., Zhai, B., Bernstein, K., Blaauw, D., Bryant, A., Chang, L., et al. "Ultralow-voltage, minimum-energy CMOS". *IBM Journal of Research and Development*. 2006;50(4.5):469-90.

[2] Paradiso, J. A., Starner, T. "Energy scavenging for mobile and wireless electronics". *Pervasive Computing, IEEE*. 2005;4(1):18-27.

[3] Jiin-Jang, M., Ching-Yuan, W. "A new constant-field scaling theory for MOSFET's". *Electron Devices, IEEE Transactions on*. 1995;42(7):1262-8.

[4] Haron, N. Z., Hamdioui, S. "Why is CMOS scaling coming to an END?". Design and Test Workshop, 2008 IDT 2008 3rd International; 2008 20-22 Dec. 2008.

[5] Golshan, K. *Physical Design Essentials: An ASIC Design Implementation Perspective*. Boston, MA: Springer US; 2007.

[6] Piguet, C. *Low-power CMOS circuits: technology, logic design and CAD tools*. Boca Raton, FL: CRC/Taylor & Francis; 2006. 440 p.

[7] Blesken, M., Lütkemeier, S., Rückert, U. "Multiobjective optimization for transistor sizing sub-threshold CMOS logic standard cells ". *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* 2010(May 30 2010-June 2 2010):1480 - 3

[8] Moore, G. E. "No exponential is forever: but "Forever" can be delayed! [semiconductor industry]". Solid-State Circuits Conference, 2003 Digest of Technical Papers ISSCC 2003 IEEE International; 2003.

[9] Skotnicki, T., Hutchby, J. A., King, T.-J., Philip Wong, H.-S., Boeuf, F. "The end of CMOS Scaling". IEEE Circuits & Devices Magazine. 2005.

[10] Dreslinski, R. G., Wieckowski, M., Blaauw, D., Sylvester, D., Mudge, T. "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits". *Proceedings of the IEEE*. 2010;98(2):253-66.

[11] Hongning, Y., Macary, V., Huber, J. L., Won-Gi, M., Baird, B., Jiangkai, Z. "Current mismatch due to local dopant fluctuations in MOSFET channel". *Electron Devices, IEEE Transactions on*. 2003;50(11):2248-54.

[12] Rithe, R., Chou, S., Jie, G., Wang, A., Datla, S., Gammie, G., et al. "Cell Library Characterization at Low Voltage Using Non-linear Operating Point Analysis of Local Variations". VLSI Design (VLSI Design), 2011 24th International Conference on; 2011 2-7 Jan. 2011.

[13] Calhoun, B. H., Brooks, D. "Can Subthreshold and Near-Threshold Circuits Go Mainstream?". *Micro, IEEE*. 2010;30(4):80-5.

[14] Meinerzhagen, P., Andersson, O., Sherazi, Y., Burg, A., Rodrigues, J. "Synthesis strategies for sub-Vt systems". Circuit Theory and Design (ECCTD), 2011 20th European Conference on; 2011 29-31 Aug. 2011.

[15] Vittoz, E. A. "Weak Inversion for Ultimate Low-Power Logic". In: Piguet C. *Low-power CMOS circuits: technology, logic design and CAD tools*. Boca Raton, FL: CRC/Taylor & Francis; 2006.

[16] Amarchinta, S., Kudithipudi, D. "Ultra low energy standard cell design optimization for performance and placement algorithm". Green Computing Conference, 2010 International; 2010 15-18 Aug. 2010.

[17] Berge, H. K. O., Aunet, S. "Multi-objective optimization of minority-3 functions for ultra-low voltage supplies". Circuits and Systems (ISCAS), 2011 IEEE International Symposium on; 2011 15-18 May 2011.

[18] Markovic, D., Wang, C. C., Alarcon, L. P., Tsung-Te, L., Rabaey, J. M. "Ultralow-Power Design in Near-Threshold Region". *Proceedings of the IEEE*. 2010;98(2):237-52.

[19] Borah, M., Owens, R. M., Irwin, M. J. "Transistor sizing for low power CMOS circuits". *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 1996;15(6):665-71.

[20] Weste, N. H. E., Eshraghian, K. "Introduction to CMOS circuits". *Principles of CMOS VLSI design: a systems perspective*. 2 ed. Reading, Mass.: Addison-Wesley; 1993. p. 3-40.

[21] Weste, N. H. E., Harris, D. M. *Integrated circuit design*. 4 ed. Boston, Mass.: Pearson; 2011. 751 p.

[22] Weste, N. H. E., Eshraghian, K. *Principles of CMOS VLSI design: a systems perspective*. Reading, Mass.: Addison-Wesley; 1993. 713 p.

[23] Naik, S., Chandel, R. "Design of a Low Power Flip-Flop Using CMOS Deep Sub Micron Technology". Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on; 2010 12-13 March 2010.

[24] Granhaug, K. *Reducing Power Dissipation while increasing Yield and Defect-Tolerance in Subthreshold CMOS*: University of Oslo; 2006.

[25] Sanyal, A., Rastogi, A., Wei, C., Kundu, S. "An Efficient Technique for Leakage Current Estimation in Nanoscaled CMOS Circuits Incorporating Self-Loading Effects". *Computers, IEEE Transactions on*. 2010;59(7):922-32.

[26] Qi, X., Lo, S. C., Gyure, A., Luo, Y., Shahram, M., Singhal, K., et al. "Efficient subthreshold leakage current optimization - Leakage current optimization and layout migration for 90- and 65- nm ASIC libraries". *Circuits and Devices Magazine, IEEE*. 2006;22(5):39-47.

[27] Van Sickel, J. H., Venkatesh, P., Lee, K. Y. "Analysis of the pareto front of a multi-objective optimization problem for a fossil fuel power plant". Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE; 2008 20-24 July 2008.

[28] Blesken, M., Ruckert, U., Steenken, D., Witting, K., Dellnitz, M. "Multiobjective optimization for transistor sizing of CMOS logic standard cells using set-oriented numerical techniques". NORCHIP, 2009; 2009 16-17 Nov. 2009.

[29] Shams, A. M., Darwish, T. K., Bayoumi, M. A. "Performance analysis of low-power 1-bit CMOS full adder cells". *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*. 2002;10(1):20-9.

[30] Croon, J. A., Decoutere, S., Sansen, W., Maes, H. E. "Physical modeling and prediction of the matching properties of MOSFETs". Solid-State Device Research conference, 2004 ESSDERC 2004 Proceeding of the 34th European; 2004 21-23 Sept. 2004.

[31] Rui-kai, D., Jian-xin, L., Chun-li, D., Lei, L., Bin, Y. "A Parallel Monte Carlo Simulation on Cluster System for Particle Transport". Artificial Intelligence and Computational Intelligence, 2009 AICI '09 International Conference on; 2009 7-8 Nov. 2009.

[32] Kim, T.-H., Keane, J., Eom, H., Kim, C. H. "Utilizing Reverse Short-Channel Effect for Optimal Subthreshold Circuit Design". *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*. 2007;15(7):821-9.

[33] Tan, G.-X., Mao, Z.-Y. "Study on Pareto front of multi-objective optimization using immune algorithm". Machine Learning and Cybernetics, 2005 Proceedings of 2005 International Conference on; 2005 18-21 Aug. 2005.

[34] *Optimization Tool Box*. [cited 2011 02.12]; Available from: http://www.mathworks.se/help/toolbox/optim/ug/optimtool.html.

[35] Hauser, J. R. "Noise margin criteria for digital logic circuits". *Education, IEEE Transactions on*. 1993;36(4):363-8.

[36] Liu, M., Cai, M., Taur, Y. "Scaling Limit of CMOS Supply Voltage from Noise Margin Considerations". Simulation of Semiconductor Processes and Devices, 2006 International Conference on; 2006 6-8 Sept. 2006.

[37] Auvergne, D., Daga, J. M., Rezzoug, M. "Signal transition time effect on CMOS delay evaluation". *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*. 2000;47(9):1362-9.

[38] Alstad, H. P., Aunet, S. "Seven subthreshold flip-flop cells". Norchip, 2007; 2007 19-20 Nov. 2007.

[39] Coates, R. F. W., Janacek, G. J., Lever, K. V. "Monte Carlo simulation and random number generation". *Selected Areas in Communications, IEEE Journal on*. 1988;6(1):58-66.

[40] Hasanbegovic, A. *Tutorial: Digital Logic Design and Verification Flow Using a Custom Standard Cell Library*. 2011.

[41] Zwoliński, M. *Digital system design with VHDL*. Harlow: Prentice Hall; 2004. 368 p.

[42] Navi, K., Kavehei, O. "Low-Power and High-Performance 1-Bit CMOS Full-Adder Cell ". *Journal of Computers*. 2008;3:48-54.

[43] Granhaug, K., Aunet, S. "Six subthreshold full adder cells characterized in 90 nm CMOS technology". Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE; 2006 0-0 0.

[44] "Library Design Guidelines". Virtuoso Abstract Generator User Guide. 2009. p. 331-72.

[45] Glesner, M., Eveking, H., Indrusiak, L., Mooney, V., Reis, R. *VLSI-SOC: From Systems to Chips: IFIP TC 10/ WG 10.5 Twelfth International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2003), December 1-3, 2003, Darmstadt, Germany*. Boston, MA: International Federation for Information Processing; 2006.

[46] Mano, M. M. *Digital design*. 4 ed. Upper Saddle River, N.J.: Pearson Prentice-Hall; 2007. 608 p.